

The luatexbase-modutils package

Manuel Pégourié-Gonnard

mpg@elzevir.fr

Élie Roux

elie.roux@telecom-bretagne.eu

v0.2 2010-05-12

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). Functions for error reporting are provided too.

It also loads `luatexbase-loader`.

Contents

1	Documentation	1
1.1	T _E X macros	2
1.2	Lua functions	2
2	Implementation	2
2.1	T _E X package	2
2.1.1	Preliminaries	3
2.1.2	User macros	4
2.2	Lua module	5
2.3	Internal functions and data	5
2.3.1	Error, warning and info function for modules	5
2.3.2	module loading and providing functions	6
3	Test files	7

1 Documentation

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

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

to provide anything for options, but implements a system for identification and version checking similar to L^AT_EX's system. Both T_EX macros and Lua functions are provided.

This package also provides Lua functions for reporting errors, warnings, etc.

It is important to notice that Lua's standard function `module()` is completely unrelated to the present package. It has nothing to do with identification and deals only with namespaces.² So, you should continue to use it normally, unlike the `require()` function which can be replaced with this package's `luatexbase.require_module()`.

1.1 T_EX macros

The two macros `\luatexUseModule` and `\luatexRequireModule` are very similar and are interfaces to the Lua functions `use_module` and `require_module`. The only difference between those macros is the number of arguments (just as the underlying Lua functions): `\luatexUseModule` only take one argument: the module name³ while `\luatexRequireModule` takes another argument for specifying a minimal version (see below). With L^AT_EX, this argument is the first and is optional. Otherwise, it's the second one and it's mandatory.

1.2 Lua functions

The main functions are `luatexbase.require_module` and `luatexbase.use_module` which may be used as a replacement to `require()`. The only difference between these functions is, `require_module` accepts a second, optional argument in order to specify a minimal version. They do the same as `require()` but also make sure the module loaded correctly identifies itself with the name given, and its version is greater than the minimal version required. The version can be given either as a (floating point) number or as a date in YYYY/MM/DD format.

Modules identify themselves using `luatexbase.provides_module`, whose only argument is a table with some information about the module. The mandatory fields are `name` (a string), `version` (a number), `date` (a string) and `description` (a string). Other fields are optional and ignored, and usually include `copyright`, `author` and `license`.

Functions for reporting are provided; similarly to L^AT_EX's `\PackageError` etc. they take the module name as their first argument and include it in the printed message in an appropriate way. The remaining arguments are passed to `string.format()` before being printed.

The functions provided (all found in the `luatexbase` table) are `module_error`, `module_warning`, `module_info` (writes to terminal and log), `module_log` (writes only to the log file) and `module_term` (writes only to the terminal).

2 Implementation

2.1 T_EX package

¹ `(*texpackage)`

²More precisely, it modifies the current environment.

³without extension

2.1.1 Preliminaries

Reload protection, especially for Plain T_EX.

```
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}[2010/05/27 v0.2a Module utilities for LuaTeX]
```

Make sure LuaT_EX 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 PackageWarningNoLine\endcsname\relax
53       \def\x#1#2{\begingroup\newlinechar10
54         \immediate\write16{Package #1 warning: #2}\endgroup}
55     \else
56       \let\x\PackageWarningNoLine
57     \fi
58   \expandafter\endgroup
59   \x{luatexbase-modutils}{LuaTeX is required for this package. Aborting.}
60   \lltxb@modutils@AtEnd
61   \expandafter\endinput
62 \fi

    Load luatexbase-loader (hence luatexbase-compat) and require supporting Lua module.
63 \begingroup\expandafter\expandafter\expandafter\endgroup
64 \expandafter\ifx\csname RequirePackage\endcsname\relax
65   \input luatexbase-loader.sty
66 \else
67   \RequirePackage{luatexbase-loader}
68 \fi
69 \luatexbase@directlua{require('luatexbase.modutils')}

    Make sure the primitives we need are available.
70 \luatexbase@ensure@primitive{luaescapestring}

```

2.1.2 User macros

Interface to `use_module()`.

```

71 \def\luatexUseModule#1{\luatexbase@directlua{%
72   luatexbase.use_module("\luatexluaescapestring{#1}")}}

```

Interface to `require_module()` with syntax depending on the format.

```

73 \begingroup\expandafter\expandafter\expandafter\endgroup
74 \expandafter\ifx\csname newcommand\endcsname\relax
75   \def\luatexRequireModule#1#2{%
76     \luatexbase@directlua{luatexbase.require_module(
77       "\luatexluaescapestring{#1}", "\luatexluaescapestring{#2}")}}
78 \else
79   \newcommand\luatexRequireModule[2][0]{%
80     \luatexbase@directlua{luatexbase.require_module(
81       "\luatexluaescapestring{#2}", "\luatexluaescapestring{#1}")}}
82 \fi
83 \lltxb@modutils@AtEnd
84 \end{texpackage}

```

2.2 Lua module

```
85 (*luamodule)
86 module("luatexbase", package.seeall)
```

2.3 Internal functions and data

Tables holding informations about the modules loaded and the versions required.

```
87 local modules = modules or {}
88 local requiredversions = {}
```

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

```
89 local function datetonenumber(date)
90     numbers = string.gsub(date,("(%d+)/(%d+)/(%d+)", "%1%2%3")
91     return tonumber(numbers)
92 end
```

Say if a string is a date in YYYY/MM/DD format.

```
93 local function isdate(date)
94     for _, _ in string.gmatch(date, "%d+/%d+/%d+") do
95         return true
96     end
97     return false
98 end
```

Parse a version into a table indicating a type (date or number), a numeric version and the original version string.

```
99 local date, number = 1, 2
100 local function parse_version(version)
101     if isdate(version) then
102         return {type = date, version = datetonenumber(version), orig = version}
103     else
104         return {type = number, version = tonumber(version), orig = version}
105     end
106 end
```

2.3.1 Error, warning and info function for modules

Here are the reporting functions for the modules. For errors, Lua's `error()` is used. For now, the error reports look less good than with $\mathrm{T}_{\mathrm{E}}\mathrm{X}$'s `\errmessage`, but hopefully it will be improved in future versions of $\mathrm{LuaT}_{\mathrm{E}}\mathrm{X}$. We could invoke `\errmessage` using `tex.sprint()`, but it may cause problems on the $\mathrm{T}_{\mathrm{E}}\mathrm{X}$ end, and moreover `error()` will still be used by Lua for other errors, so it makes messages more consistent.

```
107 local function module_error_int(mod, ...)
108     error('Module '..mod..' error: '..string.format(...), 3)
109 end
110 function module_error(mod, ...)
111     module_error_int(mod, ...)
112 end
113 function module_warning(mod, ...)
```

```

114 texio.write_nl("Module "..mod.." warning: "..string.format(...))
115 end
116 function module_info(mod, ...)
117 texio.write_nl(mod..": "..string.format(...))
118 end
119 function module_log(mod, ...)
120 texio.write_nl('log', mod..": "..string.format(...))
121 end
122 function module_term(mod, ...)
123 texio.write_nl('term', mod..": "..string.format(...))
124 end

```

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

```

125 local function err(...) module_error_int('luatexbase.modutils', ...) end
126 local function warn(...) module_warning('luatexbase.modutils', ...) end

```

2.3.2 module loading and providing functions

Load a module without version checking.

```

127 function use_module(name)
128 require(name)
129 if not modules[name] then
130 warn("Module didn't properly identified itself: %s", name)
131 end
132 end

```

Load a module with optional version checking.

```

133 function require_module(name, version)
134 if not version then
135 return use_module(name)
136 end
137 luaversion = parse_version(version)
138 if modules[name] then
139 if luaversion.type == date then
140 if datetotnumber(modules[name].date) < luaversion.version then
141 err("found module '%s' loaded in version %s, "
142 .."but version %s was required",
143 name, modules[name].date, version)
144 end
145 else
146 if modules[name].version < luaversion.version then
147 err("found module '%s' loaded in version %.02f, "
148 .."but version %s was required",
149 name, modules[name].version, version)
150 end
151 end
152 else
153 requiredversions[name] = luaversion
154 use_module(name)
155 end

```

156 end

Provide identification information for a module.

```
157 function provides_module(mod)
158   if not mod then
159     err('cannot provide nil module')
160     return
161   end
162   if not mod.version or not mod.name or not mod.date
163   or not mod.description then
164     err("invalid module registered: "
165     .."fields name, version, date and description are mandatory")
166     return
167   end
168   requiredversion = requiredversions[mod.name]
169   if requiredversion then
170     if requiredversion.type == date
171     and requiredversion.version > datetonenumber(mod.date) then
172       err("loading module %s in version %s, "
173       .."but version %s was required",
174       mod.name, mod.date, requiredversion.orig)
175     elseif requiredversion.type == number
176     and requiredversion.version > mod.version then
177       err("loading module %s in version %.02f, "
178       .."but version %s was required",
179       mod.name, mod.version, requiredversion.orig)
180     end
181   end
182   modules[mod.name] = mod
183   texio.write_nl('log', string.format("Lua module: %s %s v%.02f %s\n",
184   mod.name, mod.date, mod.version, mod.description))
185 end
186 </luamodule>
```

3 Test files

A dummy lua file for tests.

```
187 <testdummy>
188 luatexbase.provides_module {
189   name      = 'test-modutils',
190   date       = '2000/01/01',
191   version    = 1,
192   description = 'dummy test package',
193 }
194 </testdummy>
```

We just check that the package loads properly, under both LaTeX and Plain TeX. Anyway, the test files of other modules using this one already are a test...

```
195 <testplain>\input luatexbase-modutils.sty
196 <testlatex>\RequirePackage{luatexbase-modutils}
197 <*testplain,testlatex>
198 \luatexUseModule{test-modutils}
199 \luatexRequireModule
200 <testlatex>[1970/01/01]
201 {test-modutils}
202 <testplain>{1970/01/01}
203 </testplain,testlatex>
204 <testplain>\bye
205 <testlatex>\stop
```