The luatexbase-modutils package

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

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

v0.6 2013-05-11

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 **2**
- 1.1 Scope of this package 2
- 1.2 TeX macros 2
- 1.3 Lua functions 2
- 1.4 Templates 3

2 Implementation **4**
- 2.1 TeX 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.

[1] and 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 unction `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 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.)

```
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 TEX package

```
⟨*texpackage⟩
```

2.1.1 Preliminaries

Catcode defenses and reload protection.

```
\begingroup\catcode61\catcode48\catcode32=10\relax% = and space
  \catcode123 1 % {
  \catcode125 2 % }
  \catcode 35 6 % #
  \toks0\expandafter{\expandafter\endlinechar\the\endlinechar}%
  \edef\x{\endlinechar13}%
  \def\y#1 #2 {%
    \toks0\expandafter{\the\toks0 \catcode#1 \the\catcode#1}%
    \edef\x{\x \catcode#1 #2}}%
  \y  13  5 % carriage return
  \y  61 12 % =
  \y  32 10 % space
  \y 123  1 % {
  \y 125  2 % }
  \y  35  6 % #
  \y  64 11 % @ (letter)
  \y  10 12 % new line ^^J
  \y  34 12 % "
  \y  39 12 % '
  \y  40 12 % (
  \y  41 12 % )
  \y  44 12 % ,
  \y  45 12 % -
  \y  46 12 % .
  \y  47 12 % /
  \y  58 12 % :
  \y  91 12 % [
  \y  93 12 % ]
  \y  94  7 % ^
  \y  95  8 % _
  \y  96 12 % `
  \toks0\expandafter{\the\toks0 \relax\noexpand\endinput}%
  \edef\y#1{\noexpand\expandafter\endgroup%
    \noexpand\ifx#1\relax \edef#1{\the\toks0}\x\relax%
    \noexpand\else \noexpand\expandafter\noexpand\endinput%
    \noexpand\fi}%
\expandafter\y\csname luatexbase@modutils@sty@endinput\endcsname%
```

Package declaration.

```
\begingroup
  \expandafter\ifx\csname ProvidesPackage\endcsname\relax
    \def\x#1[#2]{\immediate\write16{Package: #1 #2}}
  \else
    \let\x\ProvidesPackage
  \fi
\expandafter\endgroup
\x{luatexbase-modutils}[2013/05/11 v0.6 Module utilities for LuaTeX]
```

Make sure LuaTEX is used.

```
\begingroup\expandafter\expandafter\expandafter\endgroup
\expandafter\ifx\csname RequirePackage\endcsname\relax
  \input ifluatex.sty
\else
  \RequirePackage{ifluatex}
\fi
\ifluatex\else
  \begingroup
    \expandafter\ifx\csname PackageError\endcsname\relax
      \def\x#1#2#3{\begingroup \newlinechar10
        \errhelp{#3}\errmessage{Package #1 error: #2}\endgroup}
    \else
      \let\x\PackageError
    \fi
  \expandafter\endgroup
  \x{luatexbase-modutils}{LuaTeX is required for this package. Aborting.}{%
    This package can only be used with the LuaTeX engine^^J%
    (command `lualatex' or `luatex').^^J%
    Package loading has been stopped to prevent additional errors.}
  \expandafter\luatexbase@modutils@sty@endinput%
\fi
```

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

```
\expandafter\ifx\csname RequirePackage\endcsname\relax
  \input luatexbase-loader.sty
\else
```

```
  \RequirePackage{luatexbase-loader}
\fi
\luatexbase@directlua{require('luatexbase.modutils')}
\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.

```
\ifdefined\kernel@ifnextchar
  \let\lltxb@ifnextchar\kernel@ifnextchar
\else
  \chardef\lltxb@zero0
  \chardef\lltxb@two2
  \long\def\lltxb@ifnextchar#1#2#3{%
    \begingroup
    \let\lltxb@CharToken= #1\relax
    \toks\lltxb@zero{#2}%
    \toks\lltxb@two{#3}%
    \futurelet\lltxb@LetToken\lltxb@ifnextchar@
  }
  \def\lltxb@ifnextchar@{%
    \ifx\lltxb@LetToken\lltxb@CharToken
      \expandafter\endgroup\the\toks\expandafter\lltxb@zero
    \else
      \ifx\lltxb@LetToken\lltxb@SpaceToken
         \expandafter\expandafter\expandafter\lltxb@@ifnextchar
      \else
         \expandafter\endgroup\the\toks
         \expandafter\expandafter\expandafter\lltxb@two
      \fi
    \fi
  }
  \begingroup
    \def\x#1{\endgroup
      \def\lltxb@@ifnextchar#1{%
         \futurelet\lltxb@LetToken\lltxb@ifnextchar@
      }%
    }%
  \x{ }
  \begingroup
    \def\x#1{\endgroup
      \let\lltxb@SpaceToken= #1%
    }%
  \x{ }
\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).

```
\def\RequireLuaModule#1{%
```

```
  \lltxb@ifnextchar[{\lltxb@requirelua{#1}}{\lltxb@requirelua{#1}[]}}
\def\lltxb@requirelua#1[#2]{%
  \luatexbase@directlua{luatexbase.require_module(
    "\luatexluaescapestring{#1}"
    \expandafter\ifx\expandafter\/\detokenize{#2}\/\else
      , "\luatexluaescapestring{#2}"
    \fi)}}
```

```
\luatexbase@modutils@sty@endinput%
⟨/texpackage⟩
```

2.3 Lua module

```
⟨*luamodule⟩
luatexbase         = luatexbase or { }
local luatexbase   = luatexbase
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()`.

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

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

```
local function date_to_int(date)
    if date == '' then return -1 end
    local numbers = string_gsub(date, "(%d+)/(%d+)/(%d+)", "%1%2%3")
    return tonumber(numbers)
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()`.

```
local function msg_format(msg_type, mod_name, ...)
  local cont = '('..mod_name..')' .. ('Module: '..msg_type):gsub('.', ' ')
  return 'Module '..mod_name..' '..msg_type..': '
    .. string.format(...):gsub('\n', '\n'..cont) .. '\n'
end
local function module_error_int(mod, ...)
  error(msg_format('error', mod, ...), 3)
end
local function module_error(mod, ...)
  module_error_int(mod, ...)
end
luatexbase.module_error = module_error
```

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

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

```
end
luatexbase.module_warning = module_warning
local function module_info(mod, ...)
  for _, line in ipairs(msg_format('info', mod, ...):explode('\n')) do
    texio.write_nl(line)
  end
end
luatexbase.module_info = module_info
```

No line splitting or advanced formating here.

```
local function module_log(mod, msg, ...)
  texio.write_nl('log', mod..': '..msg:format(...))
end
luatexbase.module_log = module_log
```

Produce custom versions of the reporting functions.

```
local function errwarinf(name)
  return function(...) module_error_int(name, ...) end,
    function(...) module_warning(name, ...) end,
    function(...) module_info(name, ...) end,
    function(...) module_log(name, ...) end
end
luatexbase.errwarinf = errwarinf
```

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

```
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.

```
local function require_module(name, req_date)
    require(name)
    local info = modules[name]
    if not info then
        warn("module '%s' was not properly identified", name)
    elseif req_date and info.date then
        if date_to_int(info.date) < date_to_int(req_date) then
            warn("module '%s' required in version '%s'\n"
            .. "but found in version '%s'", name, req_date, info.date)
        end
    end
end
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()`.

```
local function provides_module(info)
    if not (info and info.name) then
        err('provides_module: missing information')
    end
    texio.write_nl('log', string.format("Lua module: %s %s %s %s\n",
    info.name, info.date or '', info.version or '', info.description or ''))
    modules[info.name] = info
    return errwarinf(info.name)
end
luatexbase.provides_module = provides_module
```

2.4.3 module availability and version checking

A simple table copy function.

```
local fastcopy
fastcopy = table.fastcopy or function(old)
    if old then
        local new = { }
        for k,v in next, old do
            if type(v) == "table" then
                new[k] = fastcopy(v)
            else
                new[k] = v
            end
        end
        local mt = getmetatable(old)
        if mt then
            setmetatable(new,mt)
        end
        return new
    else
        return { }
    end
end
```

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

```
local function get_module_info(name)
    local module_table = modules[name]
    if not module_table then
      return nil
    else
      return fastcopy(module_table)
    end
end
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.

```
function get_module_version(name)
    local module_table = modules[name]
    if not module_table then
      return nil
    else
      return module_table.version
    end
end
luatexbase.get_module_version = get_module_version
```

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

```
function get_module_date(name)
    local module_table = modules[name]
    if not module_table then
      return nil
    else
      return module_table.date
    end
end
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 formated as `yyyymmdd`.

```
function get_module_date_int(name)
    local module_table = modules[name]
    if not module_table then
      return nil
    else
      return module_table.date and date_to_int(module_table.date)
    end
end
luatexbase.get_module_date_int = get_module_date_int
```

Returns true if the module is loaded, false otherwise.

```
function is_module_loaded(name)
    if modules[name] then
        return true
    else
        return false
    end
end
luatexbase.is_module_loaded = is_module_loaded
```

We provide the module, for version checking.

```
provides_module({
  name        = 'luatexbase-modutils',
  date        = '2013/05/11',
  version     = 0.6,
  description = 'Module utilities for LuaTeX',
})
```

⟨/luamodule⟩

3 Test files

A dummy lua file for tests.

⟨*testdummy⟩

```
local err, warn, info, log = luatexbase.provides_module {
  name        = 'test-modutils',
  date        = '2000/01/01',
  version     = 1,
  description = 'dummy test package',
}
luatexbase.provides_module {
  name        = 'test-modutils2',
  date        = '',
  version     = 1,
  description = 'dummy test package',
}
info('It works!\nOh, rly?\nYeah rly!')
log("I'm a one-line info.")
info("1 = "..luatexbase.get_module_version('test-modutils'))
if is_module_loaded('test-modutils') then
  info("ok!")
else
  err("problem!")
end
info("2000/01/01 = "..luatexbase.get_module_info('test-modutils').date)
info("20000101 = "..luatexbase.get_module_date_int('test-modutils'))
info("-1 = "..luatexbase.get_module_date_int('test-modutils2'))
⟨/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.

```
⟨testplain⟩\input luatexbase-modutils.sty
⟨testlatex⟩\RequirePackage{luatexbase-modutils}
⟨*testplain, testlatex⟩
\RequireLuaModule{test-modutils}
\RequireLuaModule{test-modutils}[1970/01/01]
⟨/testplain, testlatex⟩
⟨testplain⟩\bye
⟨testlatex⟩\stop
```