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 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). Functions for error reporting are provided too.

It also loads luatexbase-loader.

Contents

1 Documentation

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

[1] and 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 LaTeX's system. Both TeX 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 TeX 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 name[3] while `\luatexRequireModule` takes another argument for specifying a minimal version (see below). With LaTeX, 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 LaTeX'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 TeX package

1 ⟨*texpackage⟩

[2] More precisely, it modifies the current environment.

[3] without extension

2.1.1 Preliminaries

Reload protection, especially for Plain TEX.

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

Catcode defenses.

```
\begingroup
  \catcode123 1 % {
  \catcode125 2 % }
  \catcode 35 6 % #
  \toks0{}%
  \def\x{}%
  \def\y#1 #2 {%
    \toks0\expandafter{\the\toks0 \catcode#1 \the\catcode#1}%
    \edef\x{\x \catcode#1 #2}}%
  \y 123 1  % {
  \y 125 2  % }
  \y  35 6  % #
  \y  10 12 % ^^J
  \y  34 12 % "
  \y  36 3  % $ $
  \y  39 12 % ’
  \y  40 12 % (
  \y  41 12 % )
  \y  42 12 % *
  \y  43 12 % +
  \y  44 12 % ,
  \y  45 12 % -
  \y  46 12 % .
  \y  47 12 % /
  \y  60 12 % <
  \y  61 12 % =
  \y  64 11 % @ (letter)
  \y  62 12 % >
  \y  95 12 % _ (other)
  \y  96 12 % ‘
  \edef\y#1{\endgroup\edef#1{\the\toks0\relax}\x}%
\expandafter\y\csname lltxb@modutils@AtEnd\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}[2010/05/12 v0.2 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 PackageWarningNoLine\endcsname\relax
      \def\x#1#2{\begingroup\newlinechar10
        \immediate\write16{Package #1 warning: #2}\endgroup}
    \else
      \let\x\PackageWarningNoLine
    \fi
  \expandafter\endgroup
  \x{luatexbase-modutils}{LuaTeX is required for this package. Aborting.}
  \lltxb@modutils@AtEnd
  \expandafter\endinput
\fi
```

Load luatexbase-loader (hence luatexbase-compat) and require supporting Lua module.

```
\begingroup\expandafter\expandafter\expandafter\endgroup
\expandafter\ifx\csname RequirePackage\endcsname\relax
  \input luatexbase-loader.sty
\else
  \RequirePackage{luatexbase-loader}
\fi
\luatexbase@directlua{require('luatexbase.modutils')}
```

Make sure the primitives we need are available.

```
\luatexbase@ensure@primitive{luaescapestring}
```

2.1.2 User macros

Interface to `use_module()`.

```
\def\luatexUseModule#1{\luatexbase@directlua{%
    luatexbase.use_module("\luatexluescapestring{#1}")}}
```

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

```
\begingroup\expandafter\expandafter\expandafter\endgroup
\expandafter\ifx\csname newcommand\endcsname\relax
  \def\luatexRequireModule#1#2{%
    \luatexbase@directlua{luatexbase.require_module(
      "\luatexluaescapestring{#1}", "\luatexluaescapestring{#2}")}}
\else
  \newcommand\luatexRequireModule[2][0]{%
    \luatexbase@directlua{luatexbase.require_module(
      "\luatexluaescapestring{#2}", "\luatexluaescapestring{#1}")}}
\fi
\lltxb@modutils@AtEnd
⟨/texpackage⟩
```

2.2 Lua module

```
⟨*luamodule⟩
module("luatexbase", package.seeall)
```

2.3 Internal functions and data

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

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

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

```
local function datetonumber(date)
    numbers = string.gsub(date, "(%d+)/(%d+)/(%d+)", "%1%2%3")
    return tonumber(numbers)
end
```

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

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

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

```
local date, number = 1, 2
local function parse_version(version)
    if isdate(version) then
        return {type = date, version = datetonumber(version), orig = version}
    else
        return {type = number, version = tonumber(version), orig = version}
    end
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 TEX's `\errmessage`, but hopefully it will be improved in future versions of LuaTEX. We could invoke `\errmessage` using `tex.sprint()`, but it may cause problems on the TEX end, and moreover `error()` will still be used by Lua for other errors, so it makes messages more consistent.

```
local function module_error_int(mod, ...)
  error('Module '..mod..' error: '..string.format(...), 3)
end
function module_error(mod, ...)
  module_error_int(mod, ...)
end
function module_warning(mod, ...)
```

```
  texio.write_nl("Module "..mod.." warning: "..string.format(...))
end
function module_info(mod, ...)
  texio.write_nl(mod..": "..string.format(...))
end
function module_log(mod, ...)
  texio.write_nl('log', mod..": "..string.format(...))
end
function module_term(mod, ...)
  texio.write_nl('term', mod..": "..string.format(...))
end
```

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

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

2.3.2 module loading and providing functions

Load a module without version checking.

```
function use_module(name)
    require(name)
    if not modules[name] then
        warn("Module didn't properly identified itself: %s", name)
    end
end
```

Load a module with optional version checking.

```
function require_module(name, version)
    if not version then
        return use_module(name)
    end
    luaversion = parse_version(version)
    if modules[name] then
        if luaversion.type == date then
            if datetonumber(modules[name].date) < luaversion.version then
                err("found module `%s' loaded in version %s, "
                .."but version %s was required",
                name, modules[name].date, version)
            end
        else
            if modules[name].version < luaversion.version then
                err("found module `%s' loaded in version %.02f, "
                .."but version %s was required",
                name, modules[name].version, version)
            end
        end
    else
        requiredversions[name] = luaversion
        use_module(name)
    end
```

```
end
```

Provide identification information for a module.

```
function provides_module(mod)
    if not mod then
        err('cannot provide nil module')
        return
    end
    if not mod.version or not mod.name or not mod.date
    or not mod.description then
        err("invalid module registered: "
        .."fields name, version, date and description are mandatory")
        return
    end
    requiredversion = requiredversions[mod.name]
    if requiredversion then
        if requiredversion.type == date
        and requiredversion.version > datetonumber(mod.date) then
            err("loading module %s in version %s, "
            .."but version %s was required",
            mod.name, mod.date, requiredversion.orig)
        elseif requiredversion.type == number
        and requiredversion.version > mod.version then
            err("loading module %s in version %.02f, "
            .."but version %s was required",
            mod.name, mod.version, requiredversion.orig)
        end
    end
    modules[mod.name] = module
    texio.write_nl('log', string.format("Lua module: %s %s v%.02f %s\n",
    mod.name, mod.date, mod.version, mod.description))
end
```

⟨/luamodule⟩

3 Test files

A dummy lua file for tests.

```
⟨*testdummy⟩
luatexbase.provides_module {
  name        = 'test-modutils',
  date        = '2000/01/01',
  version     = 1,
  description = 'dummy test package',
}
⟨/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...

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