

The `luacode` package

Manuel Pégourié-Gonnard <mpg@elzevir.fr>

v1.0 2010/11/08

Abstract

Executing Lua code from within \TeX with `\directlua` can sometimes be tricky: there is no easy way to use the percent character, counting backslashes may be hard, and Lua comments don't work the way you expect. This package provides the `\luaexec` command and the `luacode(*)` environments to help with these problems.

Contents

1	Documentation	1
2	Implementation	2
2.1	Preliminaries	2
2.2	Internal code	3
2.3	Public macros and environments	5
3	Test file	6

1 Documentation

For an introduction to the most important gotchas of `\directlua`, see `lualatex-doc.pdf`. Before presenting the tools in this package, let me insist that the best way to manage a non-trivial piece of Lua code is probably to use an external file and source it from Lua, as explained in the cited document.

First, the exact syntax of `\directlua` has changed along version of Lua \TeX , so this package provides a `\luadirect` command which is an exact synonym of `\directlua` except that it doesn't have the funny, changing parts of its syntax.¹

The problems with `\directlua` (or `\luadirect`) are mainly with \TeX special characters. Actually, things are not that bad, since most special characters do work, namely: `_`, `^`, `&`, `$`, `{`, `}`. Three are a bit tricky but they can be managed with `\string`: `\` and `~`. Only `%` is really hard to obtain. Also, \TeX macros are expanded, which is good since it allows to pass information from \TeX to Lua, but you must be careful and use only purely expandable macros.

The `\luaexec` command is similar to `\luadirect` but with a few additional features:² `\%` gives a double backslash (convenient in Lua strings), `\%` a percent character, and `~` just works. If you happen to need a single backslash, you can still use a construct like `\string\n` just as with `\luadirect`. Also, \TeX macros are expanded.

¹And it expands in two steps. If you don't know what it means, then you hopefully don't need to.

²And one major drawback: it is not purely expandable. See above.

The `luacode` environment is similar, except that you can use `%` and other special characters directly (but `\%` also works). Only the backslash and the braces keep their special meaning, so that macros still work as usual, and you still need to use a `\string` trick to get a single backslash.

A variant `luacode*` goes further and makes even backslash a normal character, so that you don't need any trick to obtain a single backslash. On the other end, macros don't work any more.

In addition, the line breaks in the environments are respected, so that you can use line-wise Lua comments in the normal way, without mistakenly commenting the rest of the chunk.

	<code>\luadirect</code>	<code>\luaexec</code>	<code>luacode</code>	<code>luacode*</code>
Macros	Yes	Yes	Yes	No
Single backslash	Using <code>\string</code>	Using <code>\string</code>	Using <code>\string</code>	Just <code>\</code>
Double backslash	<code>\string\\</code>	<code>\\</code>	<code>\\</code>	<code>\\</code>
Tilde	<code>\string~</code>	<code>~</code>	<code>~</code>	<code>~</code>
Sharp	<code>\string#</code>	<code>\#</code>	<code>\#</code> or <code>#</code>	<code>#</code>
Percent	No easy way	<code>\%</code>	<code>\%</code> or <code>%</code>	<code>%</code>
TeX line comments	Yes	Yes	No	No
Lua line comments	No	No	Yes	Yes

Note that you cannot use the environments inside the argument of a command (just as with verbatim commands and environments). If you wish to define derived environments, you'll need to use `\luacode ... \endluacode` instead of the usual `\begin \end` pair in your environment's definition. For the starred variant, use `\luacodestar` and `\endluacodestar`.

The test file (section 3, or `test-luacode.tex` in the same directory as this document) provides stupid but complete examples.

2 Implementation

```
1 (*texpackage)
```

2.1 Preliminaries

Catcode defenses.

```
2 \begingroup\catcode61\catcode48\catcode32=10\relax% = and space
3 \catcode123 1 % {
4 \catcode125 2 % }
5 \catcode 35 6 % #
6 \toks0{\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 39 12 % '

```

```

19 \y 40 12 % (
20 \y 41 12 % )
21 \y 42 12 % *
22 \y 45 12 % -
23 \y 46 12 % .
24 \y 47 12 % /
25 \y 91 12 % [
26 \y 93 12 % ]
27 \y 94 7 % ^
28 \y 96 12 % ‘
29 \y 126 13 % ~
30 \toks0\expandafter{\the\toks0 \relax\noexpand\endinput}%
31 \edef\y#1{\noexpand\expandafter\endgroup%
32 \noexpand\ifx#1\relax \edef#1{\the\toks0}\x\relax%
33 \noexpand\else \noexpand\expandafter\noexpand\endinput%
34 \noexpand\fi}%
35 \expandafter\y\csname luacode@sty@endinput\endcsname%

```

Package declaration.

```
36 \ProvidesPackage{luacode}[2011/05/24 v1.0 lua-in-tex helpers (mpg)]
```

Make sure LuaTeX is used.

```

37 \RequirePackage{ifluatex}
38 \ifluatex\else
39 \PackageError{luacode}{LuaTeX is required for this package. Aborting.}{%
40 This package can only be used with the LuaTeX engine\MessageBreak
41 (command ‘lualatex’). Package loading has been stopped\MessageBreak
42 to prevent additional errors.}
43 \expandafter\luacode@sty@endinput
44 \fi

```

Use luatexbase for catcode tables.

```
45 \RequirePackage{luatexbase}
```

2.2 Internal code

Execute a piece of code, with shortcuts for double-backslash, percent and tilde, and trying to preserve newlines. This internal macro is long so that we can use in the environment, while the corresponding user command will be short. Make sure ~ is active.

```

46 \begingroup \catcode'\~\active \expandafter\endgroup
47 \@firstofone{%
48 \newcommand*\luacode@execute [1] {%
49 \begingroup
50 \escapechar92
51 \newlinechar10
52 \edef\\{\string\\}%
53 \edef~{\string~}%
54 \let\%=\luacode@percentchar
55 \let\#=\luacode@sharpchar
56 \expandafter\expandafter\expandafter\endgroup
57 \luatexbase@directlua{#1}}
58 }

```

Catcode 12 percent and sharp characters for use in the previous command.

```
59 \begingroup \escapechar\m@ne \edef\aux{\endgroup
60 \unexpanded{\newcommand\luacode@percentchar}{\string\}%}
61 \unexpanded{\newcommand\luacode@sharpchar }{\string\#}%}
62 }\aux
```

Generic code for environments; the argument is the name of a catcode table. We're normally inside a group, but let's open a new one in case we're called directly rather than using `\begin`. Define the end marker to be `\end{<envname>}` with current catcodes.

```
63 \newcommand*\luacode@begin [1] {%
64 \begingroup
65 \escapechar92
66 \luatexcatcodetable#1\relax
67 \edef\luacode@endmark{\string\end{\@currenvir}}%
68 \expandafter\def \expandafter\luacode@endmark \expandafter{%
69 \luatexscantextokens \expandafter{\luacode@endmark}}%
70 \luacode@grab@body}
```

We'll define the body grabber in a moment, but let's see how the environment ends now.

```
71 \newcommand\luacode@end{%
72 \edef\luacode@next{%
73 \noexpand\luacode@execute{\the\luacode@lines}%
74 \noexpand\end{\@currenvir}}%
75 \expandafter\endgroup
76 \luacode@next}
```

It is not possible to grab the body using a macro with delimited argument, since the end marker may contain open-group characters, depending on the current catcode regime. So we collect it linewise and check each line against the end marker.

Storage for lines.

```
77 \newtoks\luacode@lines
78 \newcommand*\luacode@addline [1] {%
79 \luacode@lines\expandafter{\the\luacode@lines^^J#1}}
```

Loop initialisation. Set `endlinechar` explicitly so that we can use it as a delimiter (and later when writing the code to Lua).

```
80 \newcommand \luacode@grab@body {
81 \luacode@lines{}%
82 \endlinechar10
83 \luacode@grab@lines}
```

The actual line-grabbing loop.

```
84 \long\def\luacode@grab@lines#1^^J{%
85 \def\luacode@curr{#1}%
86 \luacode@strip@spaces
87 \ifx\luacode@curr\luacode@endmark
88 \expandafter\luacode@end
89 \else
90 \expandafter\luacode@addline\expandafter{\luacode@curr}%
91 \expandafter\luacode@grab@lines
92 \fi}
```

Strip catcode 12 spaces from the beginning the token list inside `\luacode@curr`. First we need catcode 12 space, then we procede in the usual way.

```

93 \begingroup\catcode32 12 \expandafter\endgroup
94 \@firstofone{\newcommand\luacode@spaceother{ }}
95 \newcommand \luacode@strip@spaces {%
96   \expandafter\luacode@strip@sp@peek\luacode@curr\@nil}
97 \newcommand \luacode@strip@sp@peek {%
98   \futurelet\@let@token\luacode@strip@sp@look}
99 \newcommand \luacode@strip@sp@look {%
100  \expandafter\ifx\luacode@spaceother\@let@token
101   \expandafter\@firstoftwo
102  \else
103   \expandafter\@secondoftwo
104  \fi{%
105   \afterassignment\luacode@strip@sp@peek
106   \let\@let@token=
107  }{%
108   \luacode@strip@sp@def
109  }}
110 \@ifdefinable \luacode@strip@sp@def \relax
111 \def \luacode@strip@sp@def #1\@nil{%
112  \def\luacode@curr{#1}}

```

Finally, we need a custom catcode table for the default environment: everything other, except backslash, braces and letters which retain their natural catcodes.

Be carefull about the name of the macro for setting catcode ranges which is currently changing in luatexbase. The group here doesn't matter since catcode table settings are always global.

```

113 \newluatexcatcodetable \luacode@table@soft
114 \begingroup
115 \ifdefined\SetCatcodeRange \else
116  \let\SetCatcodeRange\setcatcoderange
117 \fi
118 \setluatexcatcodetable \luacode@table@soft {%
119  \luatexcatcodetable\CatcodeTableOther
120  \catcode 92 0
121  \catcode 123 1
122  \catcode 125 2
123  \SetCatcodeRange {65}{90} {11}
124  \SetCatcodeRange {97}{122}{11}
125 }
126 \endgroup

```

2.3 Public macros and environments

The `\luadirect` and `\luaexec` macro.

```

127 \@ifdefinable\luadirect {\let\luadirect\luatexbase@directlua}
128 \newcommand*\luaexec [1] {\luacode@execute{#1}}

```

Environments using different catcode tables.

```

129 \newenvironment {luacode} {\luacode@begin\luacode@table@soft} {}
130 \newenvironment {luacode*} {\luacode@begin\CatcodeTableOther} {}
131 \newcommand \luacodestar {\@nameuse{luacode*}}
132 \def \endluacodestar {\@nameuse{endluacode*}}

```

We're already done!

```
133 \luacode@sty@endinput
134 \end{texpackage}
```

3 Test file

```
135 <testlatex>
136 \documentclass{minimal}
137 \usepackage{luacode}
138 \begin{document}
139
140 \newcommand\foo{3}
141
142 \(\
143   \luadirect{
144     texio.write_nl("Special chars: _ ^ & $ { } working.\string\n"
145     .. "Backslashes need a bit of care.\string\n"
146     .. "Sharps and tildes too: # doubled, but \string# and \string~")
147     % a tex comment: no easy way to get a %
148     tex.sprint("\string\\pi \string\\neq", tostring(math.pi))
149     % we can use TeX macros
150     tex.sprint("-", math.sqrt(\foo))
151   }
152 \)
153
154
155 \(\
156   \luaexec{
157     texio.write_nl("Special chars: _ ^ & $ { } ~ working.\string\n"
158     .. "Backslashes still need a bit of care.\string\n"
159     .. "Single sharps are easier now: \#")
160     % a tex comment: we also get a % below
161     tex.sprint("\pi \neq ", tostring(math.pi):gsub('%.', '+'))
162     % we can use TeX macros
163     tex.sprint("-", math.sqrt(\foo))
164   }
165 \)
166
167 \[
168   \begin{luacode}
169     texio.write_nl("Special chars: _ ^ & $ { } ~ # % working.\string\n"
170     .. "Only backslashes still need a bit of care.\string\n")
171     -- a lua comment: we could use \% below, too
172     tex.sprint("\pi \neq ", tostring(math.pi):gsub('%.', '+'))
173     -- we can use TeX macros
174     tex.sprint("-", math.sqrt(\foo))
175   \end{luacode}
176 \]
177
178 \[
179   \begin{luacode*}
180     texio.write_nl("Special chars: _ ^ & $ { } ~ # % \\ working.\n")
```

```

181 -- a lua comment: the backlash is doubled as in normal Lua code
182 tex.sprint("\pi \neq ", tostring(math.pi):gsub('%.', '+'))
183 -- no way to use a TeX variable here
184 \end{luacode*}
185 \]
186
187 \newenvironment{mathluacode} { \[ \luacode }{ \endluacode \] }
188 \newenvironment{mathluacode*}{ \[ \luacodestar }{ \endluacodestar \] }
189
190 \begin{mathluacode}
191 local foo = "A full line.\string\n"
192 tex.sprint("\pi \neq ", tostring(math.pi):gsub('%.', '+'))
193 -- a lua comment: we could have used \% above, too
194 tex.sprint("-", math.sqrt(\foo))
195 \end{mathluacode}
196
197 \begin{mathluacode*}
198 local foo_bar = "A full line.\n"
199 tex.sprint("\pi \neq ", tostring(math.pi):gsub('%.', '+'))
200 -- a lua comment: no way to use a TeX variable here
201 \end{mathluacode*}
202
203 \end{document}
204 </testlatex>

```