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

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 most robust way to manage a non-trivial piece of Lua code is 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 LuaTEX, 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 carefull and use only purely expandable macros.

The `\luaexec` command is similar to `\luadirect` but with a few additional features:[2] `\\` gives a double blackslash (convenient in Lua strings), `\%` a percent character, and `~` just works.

[1] And also, it expands in two steps instead of one. If you don't known what it means, then you probably don't need to.

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

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.

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.

	`\luadirect`	`\luaexec`	`luacode`	`luacode*`
Macros	Yes	Yes	Yes	No
Single backslash	Using `\string`	Using `\string`	Using `\string`	Just `\`
Double backslash	`\string\\`	`\\`	`\\`	`\\`
Tilde	`\string~`	`~`	`~`	`~`
Sharp	`\string#`	`\#`	`\#` or `#`	`#`
Percent	No easy way	`\%`	`\%` or `%`	`%`
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 stared 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
3   \catcode123 1 % {
4   \catcode125 2 % }
5   \catcode 35 6 % #
6   \toks0{}%
7   \def\x{}%
8   \def\y#1 #2 {%
9     \toks0\expandafter{\the\toks0 \catcode#1 \the\catcode#1}%
10    \edef\x{\x \catcode#1 #2}}%
11  \y 123 1  % {
12  \y 125 2  % }
13  \y  35 6  % #
14  \y  10 12 % ^^J
15  \y  34 12 % "
16  \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 luacode@AtEnd\endcsname
```

Package declaration.

```
\ProvidesPackage{luacode}[2010/11/08 v1.0 lua-in-tex helpers (mpg)]
```

Make sure LuaTEX is used.

```
\RequirePackage{ifluatex}
\ifluatex\else
  \PackageError{luacode}{LuaTeX is required for this package. Aborting.}{%
    This package can only be used with the LuaTeX engine\MessageBreak
    (command `lualatex'). Package loading has been stopped\MessageBreak
    to prevent additional errors.}
  \lltxb@core@AtEnd
  \expandafter\endinput
\fi
```

Use luatexbase for catcode tables, and ensure we have the primitives we need.

```
\RequirePackage{luatexbase}
\luatexbase@ensure@primitive{scantextokens}%
```

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.

```
\begingroup \catcode`\~\active \expandafter\endgroup
\@firstofone{%
  \newcommand*\luacode@execute [1] {%
    \begingroup
    \escapechar92
    \newlinechar10
    \edef\\{\string\\}%
    \edef~{\string~}%
    \let\%=\luacode@percentchar
    \let\#=\luacode@sharpchar
    \expandafter\expandafter\expandafter\endgroup
    \luatexbase@directlua{#1}}
}
```

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

```
\begingroup \escapechar\m@ne \edef\aux{\endgroup
  \unexpanded{\newcommand\luacode@percentchar}{\string\%}%
  \unexpanded{\newcommand\luacode@sharpchar  }{\string\#}%
}\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 that using `\begin`. Define the end marker to be `\end{<envname>}` with current catcodes.

```
\newcommand*\luacode@begin [1] {%
  \begingroup
  \escapechar92
  \luatexcatcodetable#1\relax
  \edef\luacode@endmark{\string\end{\@currenvir}}%
  \expandafter\def \expandafter\luacode@endmark \expandafter{%
    \luatexscantextokens \expandafter{\luacode@endmark}}%
  \luacode@grab@body}
```

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

```
\newcommand\luacode@end{%
  \edef\luacode@next{%
    \noexpand\luacode@execute{\the\luacode@lines}%
    \noexpand\end{\@currenvir}}%
  \expandafter\endgroup
  \luacode@next}
```

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

Sotrage for lines.

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

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

```
\newcommand \luacode@grab@body {
  \luacode@lines{}%
  \endlinechar10
  \luacode@grab@lines}
```

The actual line-grabbing loop.

```
\long\def\luacode@grab@lines#1^^J{%
  \def\luacode@curr{#1}%
  \luacode@strip@spaces
  \ifx\luacode@curr\luacode@endmark
    \expandafter\luacode@end
  \else
    \expandafter\luacode@addline\expandafter{\luacode@curr}%
    \expandafter\luacode@grab@lines
  \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.

```
\begingroup\catcode32 12 \expandafter\endgroup
\@firstofone{\newcommand\luacode@spaceother{ }}
\newcommand \luacode@strip@spaces {%
  \expandafter\luacode@strip@sp@peek\luacode@curr\@nil}
\newcommand \luacode@strip@sp@peek {%
  \futurelet\@let@token\luacode@strip@sp@look}
\newcommand \luacode@strip@sp@look {%
  \expandafter\ifx\luacode@spaceother\@let@token
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi{%
    \afterassignment\luacode@strip@sp@peek
    \let\@let@token=
  }{%
    \luacode@strip@sp@def
  }}
\@ifdefinable \luacode@strip@sp@def \relax
\def \luacode@strip@sp@def #1\@nil{%
  \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.

```
\newluatexcatcodetable \luacode@table@soft
\setluatexcatcodetable \luacode@table@soft {%
  \luatexcatcodetable\CatcodeTableOther
  \catcode 92  0
  \catcode 123 1
  \catcode 125 2
  \setcatcoderange {65}{90} {11}
  \setcatcoderange {97}{122}{11}
}
```

2.3 Public macros and environments

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

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

Environments using different catcode tables.

```
\newenvironment {luacode}  {\luacode@begin\luacode@table@soft} {}
\newenvironment {luacode*} {\luacode@begin\CatcodeTableOther}  {}
\newcommand     \luacodestar    {\@nameuse{luacode*}}
\def            \endluacodestar {\@nameuse{endluacode*}}
```

We're already done!

```
\luacode@AtEnd
⟨/texpackage⟩
```

3 Test file

```
⟨*testlatex⟩
\documentclass{minimal}
\usepackage{luacode}
\begin{document}

\newcommand\foo{3}

\(
  \luadirect{
    texio.write_nl("Special chars: _ ^ & $ { } working.\string\n"
    .. "Backslashes need a bit of care.\string\n"
    .. "Sharps and tildes too: # doubled, but \string# and \string~")
    % a tex comment: no easy way to get a %
    tex.sprint("\string\\pi \string\\neq", tostring(math.pi))
    % we can use TeX macros
    tex.sprint("-", math.sqrt(\foo))
  }
\)


\(
  \luaexec{
    texio.write_nl("Special chars: _ ^ & $ { } ~ working.\string\n"
    .. "Backslashes still need a bit of care.\string\n"
    .. "Single sharps are easier now: \#")
    % a tex comment: we also get a % below
    tex.sprint("\\pi \\neq ", tostring(math.pi):gsub('\%.', '+'))
    % we can use TeX macros
    tex.sprint("-", math.sqrt(\foo))
  }
\)

\[
  \begin{luacode}
    texio.write_nl("Special chars: _ ^ & $ { } ~ # % working.\string\n"
    .. "Only backslashes still need a bit of care.\string\n")
    -- a lua comment: we could use \% below, too
    tex.sprint("\\pi \\neq ", tostring(math.pi):gsub('%.', '+'))
    -- we can use TeX macros
    tex.sprint("-", math.sqrt(\foo))
  \end{luacode}
\]

\[
  \begin{luacode*}
    texio.write_nl("Special chars: _ ^ & $ { } ~ # % \\ working.\n")
    -- a lua comment: the backlash is doubled as in normal Lua code
    tex.sprint("\\pi \\neq ", tostring(math.pi):gsub('%.', '+'))
    -- no way to use a TeX variable here
  \end{luacode*}
\]
```

```
\newenvironment{mathluacode} { \[ \luacode     }{ \endluacode     \] }
\newenvironment{mathluacode*}{ \[ \luacodestar }{ \endluacodestar \] }

\begin{mathluacode}
  local foo = "A full line.\string\n"
  tex.sprint("\\pi \\neq ", tostring(math.pi):gsub('%.', '+'))
  -- a lua comment: we could have used \% above, too
  tex.sprint("-", math.sqrt(\foo))
\end{mathluacode}

\begin{mathluacode*}
  local foo_bar = "A full line.\n"
  tex.sprint("\\pi \\neq ", tostring(math.pi):gsub('%.', '+'))
  -- a lua comment: no way to use a TeX variable here
\end{mathluacode*}

\end{document}
```
⟨/testlatex⟩