The lt3luabridge package: Lua without LuaTeX

Vít Novotný*

Released 2022-10-24

The lt3luabridge expl3 [2] package provides support for executing Lua code in LuaTeX or any other TeX engine that exposes the shell. The package provides interfaces to plain TeX, LaTeX, and ConTeXt formats:

```
\documentclass{standalone}
\usepackage{lt3luabridge}
\begin{document}
$ 1 + 2 = \luabridgeExecute{ print(1 + 2) } $
\end{document}
```

The package was previously part of the Markdown package [1], where it has been battle-tested since 2016. Since 2022, lt3luabridge has also been available as a separate package.

1 Loading the package

Use the `\input lt3luabridge\relax` command to load the package from plain TeX, use the `\usepackage{lt3luabridge}` command to load the package from LaTeX, and use the `\usemodule[t][lt3luabridge]` command to load the package from ConTeXt.

2 Executing Lua code

The interface for executing Lua code mimics the `\lua_now:n` function from l3luatex.

`\luabridge_now:n`
`\luabridge_now:e`

New: 2022-06-26
Updated: 2022-07-31

`\luabridge_now:n {⟨token list⟩}`

The ⟨*token list*⟩ is first tokenized by TeX, which includes converting line ends to spaces in the usual TeX manner and which respects currently-applicable TeX category codes. The resulting ⟨*Lua input*⟩ is passed to the Lua interpreter for processing. Each `\luabridge_now:n` block is treated by Lua as a separate chunk. The Lua interpreter executes the ⟨*Lua input*⟩ immediately, and in an expandable manner.

Unlike `\lua_now:n`, `\luabridge_now:n` may execute ⟨*Lua input*⟩ in a separate process from TeX. Therefore, you should not interact with TeX from ⟨*Lua input*⟩ or create global variables. The only exception is the standard output produced by the `print()` Lua function like in the example at the top of this page. The standard output of `print()` will be inserted into TeX's input stream.

`\luabridgeExecute`

New: 2022-06-26
Updated: 2022-07-31

`\luabridgeExecute {⟨token list⟩}`

The `\luabridgeExecute` document command aliases the `\luabridge_now:e` function.

---

[*] E-mail: witiko@mail.muni.cz

3 Setting and getting the method to execute Lua code

There are several methods that can be used to execute Lua code. This section describes the interface that the package provides to set the preferred method or to determine which method was used.

`\g_luabridge_method_int`

`New: 2022-06-26`

This variable controls the method used to execute Lua code. The variable is set automatically when the package is loaded and changing the value of the variable afterwards has no effect. However, we can set the value of the variable before loading the package to one of the constants described below.

`\c_luabridge_method_shell_int`

`New: 2022-07-31`

Use shell escape through the `\write18` TeX command to execute Lua code.

`\c_luabridge_method_directlua_int`

`New: 2022-06-26`

Use the `\directlua` primitive of LuaTeX to execute Lua code.

4 Setting and getting the filenames of helper files

When shell escape is used to execute Lua code, several helper files are needed to shuffle around code and output. The following variables and constants are undefined when the `\directlua` primitive of LuaTeX is used to execute Lua code.

`\g_luabridge_output_dirname_str`

`New: 2022-06-26`

This variable controls the output directory that will store the helper files. The variable should be set to the same value as the `-output-directory` parameter of the TeX engine.

`\c_luabridge_default_output_dirname_str`

`New: 2022-06-26`

This constant is the default value of `\g_luabridge_output_dirname_str`.

`\g_luabridge_helper_script_filename_str`

`New: 2022-06-26`

This variable controls the filename of a helper Lua script that will be executed from the shell using the TeX Lua interpreter.

`\c_luabridge_default_helper_script_filename_str`

`New: 2022-06-26`

This constant is the default value of `\g_luabridge_helper_script_filename_str`.

`\g_luabridge_error_output_filename_str`

New: 2022-06-26

This variable controls the filename of a helper file that will contain the error output produced by the `texlua` interpreter (if any).

`\c_luabridge_default_error_output_filename_str`

New: 2022-06-26

This constant is the default value of `\g_luabridge_error_output_filename_str`.

5 Plain TEX implementation

This section contains the implementation for plain TEX using generic expl3.

```
⟨@@=luabridge⟩
⟨*generic-package⟩
\ifx\ExplSyntaxOn\undefined
  \input expl3-generic\relax
\fi
\ExplSyntaxOn
\int_const:Nn
  \c_luabridge_method_directlua_int
  { 0 }
\int_const:Nn
  \c_luabridge_method_shell_int
  { 1 }
\int_if_exist:NF
  \g_luabridge_method_int
  {
    \int_new:N
      \g_luabridge_method_int
      \sys_if_engine_luatex:TF
        {
          \int_gset_eq:NN
            \g_luabridge_method_int
            \c_luabridge_method_directlua_int
        }
        {
          \int_gset_eq:NN
            \g_luabridge_method_int
            \c_luabridge_method_shell_int
        }
  }
\msg_new:nnn
  { luabridge }
  { method-shell }
  {
    Using~shell~escape~as~the~bridging~method
  }
\msg_new:nnn
  { luabridge }
  { method-directlua }
  {
    Using~direct~Lua~access~as~the~bridging~method
  }
\msg_new:nnn
  { luabridge }
  { unknown-method }
  {
    Unknown~bridging~method:~#1
  }
\int_case:nnF
  { \g_luabridge_method_int }
  {
    { \c_luabridge_method_shell_int }
      {
        \msg_info:nn
          { luabridge }
          { method-shell }
      }
    { \c_luabridge_method_directlua_int }
      {
        \msg_info:nn
          { luabridge }
          { method-directlua }
      }
  }
  {
    \cs_generate_variant:Nn
      \msg_error:nnn
      { nnV }
    \msg_error:nnV
      { luabridge }
      { unknown-method }
      \g_luabridge_method_int
  }
\int_compare:nNnT
  { \g_luabridge_method_int }
  =
  { \c_luabridge_method_shell_int }
  {
    \str_const:Nn
      \c_luabridge_default_output_dirname_str
      { . }
    \str_const:Nx
      \c_luabridge_default_helper_script_filename_str
      { \jobname.luabridge.lua }
    \str_const:Nx
      \c_luabridge_default_error_output_filename_str
      { \jobname.luabridge.err }
    \str_if_exist:NF
      \g_luabridge_output_dirname_str
      {
        \str_new:N
          \g_luabridge_output_dirname_str
        \str_gset_eq:NN
```

```
          \g_luabridge_output_dirname_str
          \c_luabridge_default_output_dirname_str
      }
    \str_if_exist:NF
      \g_luabridge_helper_script_filename_str
      {
        \str_gset_eq:NN
          \g_luabridge_helper_script_filename_str
          \c_luabridge_default_helper_script_filename_str
      }
    \str_if_exist:NF
      \g_luabridge_error_output_filename_str
      {
        \str_gset_eq:NN
          \g_luabridge_error_output_filename_str
          \c_luabridge_default_error_output_filename_str
      }
    \cs_new:Nn
      \luabridge_now:n
      {
        \iow_open:NV
          \g_tmpa_iow
          \g_luabridge_helper_script_filename_str
        \msg_info:nnV
          { luabridge }
          { writing-helper-script }
          \g_luabridge_helper_script_filename_str
```

Escape `"` and `\` in the Lua code, so that we can represent it as a double-quoted string that we can pass into the `load()` Lua built-in and fail gracefully if the Lua code fails to compile.

```
        \tl_set:Nx
          \l_tmpa_tl
          { \tl_to_str:n { #1 } }
        \regex_replace_all:nnN
          { [\\"] }
          { \\\0 }
          \l_tmpa_tl
        \tl_set:Nx
          \l_tmpa_tl
          {
            local~ran_ok, err = pcall(function()
              local~ran_ok, kpse = pcall(require,~"kpse")
              if~ran_ok~then~kpse.set_program_name("luatex") end~
              assert(load(" \exp_not:V \l_tmpa_tl "))()
            end)
            if~not~ran_ok~then~
              local~file = io.open("
                \g_luabridge_output_dirname_str /
                \g_luabridge_error_output_filename_str
              ", "w")
              if~file~then~
                file:write(err .. " \iow_char:N \\ n ")
                file:close()
```

```
     end~
     print('
      \iow_char:N \\ \iow_char:N \\ begingroup
       \iow_char:N \\ \iow_char:N \\ ExplSyntaxOn
       \iow_char:N \\ \iow_char:N \\ csname~
       msg_error:nnvv\iow_char:N \\ \iow_char:N \\ endcsname
        { luabridge }
        { failed-to-execute }
        { g_luabridge_output_dirname_str }
        { g_luabridge_error_output_filename_str }
      \iow_char:N \\ \iow_char:N \\ endgroup
     ')
    end
   }
  \iow_now:NV
   \g_tmpa_iow
   \l_tmpa_tl
  \iow_close:N
   \g_tmpa_iow
  \msg_info:nnV
   { luabridge }
   { executing-helper-script }
   \g_luabridge_helper_script_filename_str
  \sys_get_shell:xnNTF
   {
    texlua~
     \g_luabridge_output_dirname_str /
     \g_luabridge_helper_script_filename_str
   }
   { }
   \l_tmpa_tl
   {
    \l_tmpa_tl
   }
   {
    \msg_error:nn
     { luabridge }
     { level-disabled }
   }
 }
\prg_generate_conditional_variant:Nnn
 \sys_get_shell:nnN
 { xnN }
 { TF }
\cs_generate_variant:Nn
 \msg_info:nnn
 { nnV }
\cs_generate_variant:Nn
 \msg_error:nnnn
 { nnvv }
\cs_generate_variant:Nn
 \iow_open:Nn
 { NV }
\cs_generate_variant:Nn
```

```
      \iow_now:Nn
      { NV }
    \msg_new:nnn
      { luabridge }
      { writing-helper-script }
      {
        Writing~a~helper~Lua~script~to~file~#1
      }
    \msg_new:nnn
      { luabridge }
      { executing-helper-script }
      {
        Executing~a~helper~Lua~script~from~file~#1
      }
    \msg_new:nnnn
      { luabridge }
      { failed-to-execute }
      {
        An~error~was~encountered~while~executing~Lua~code
      }
      {
        For~further~clues,~examine~file~#1 / #2
      }
    \msg_new:nnnn
      { luabridge }
      { level-disabled }
      {
        Shell~escape~seems~to~be~disabled
      }
      {
        You~may~need~to~run~TeX~with~the~--shell-escape~or~the~
        --enable-write18~flag,~or~write~shell_escape=t~in~the~
        texmf.cnf~file.
      }
  }
\int_compare:nNnT
  { \g_luabridge_method_int }
  =
  { \c_luabridge_method_directlua_int }
  {
    \cs_new:Nn
      \luabridge_now:n
      {
        \tl_set:Nn
          \l_tmpa_tl
          { #1 }
        \tl_set:Nx
          \l_tmpa_tl
          {
            _ENV = setmetatable({}, {__index = _ENV})
            local~function~print(input)
              input = tostring(input)
              local~output = {}
              for~line~in~input:gmatch("[^
                    \iow_char:N \\ r
                    \iow_char:N \\ n
                  ]+") do~
                table.insert(output, line)
              end~
              tex.print(output)
            end~
            \exp_not:V \l_tmpa_tl
          }
        \lua_now:V
          \l_tmpa_tl
      }
    \cs_generate_variant:Nn
      \lua_now:n
      { V }
  }
\cs_new_protected:Npn
  \luabridgeExecute
  #1
  {
    \luabridge_now:e
      { #1 }
  }
\cs_generate_variant:Nn
  \luabridge_now:n
  { e }
\ExplSyntaxOff
⟨/generic-package⟩
```

6 LaTeX implementation

This section contains the implementation for LaTeX.

```
⟨*latex-package⟩
\RequirePackage{expl3}
\ProvidesExplPackage
  {lt3luabridge}%
  {2022-10-24}%
  {2.0.2}%
  {An expl3 package that allows you to execute Lua code in LuaTeX or any other
   TeX engine that exposes the shell}
\input lt3luabridge\relax
⟨/latex-package⟩
```

7 ConTeXt implementation

This section contains the implementation for ConTeXt. ConTeXt MkII, MkIV, and later formats are supported.

```
⟨*context-package⟩
\writestatus{loading}{ConTeXt User Module / lt3luabridge}
\startmodule[lt3luabridge]
\unprotect
```

```
\input lt3luabridge\relax
⟨/context-package⟩
```

References

[1] Vít Novotný. *Markdown. A package for converting and rendering markdown documents inside TEX.* Version 2.15.2-0-gb238dbc. May 31, 2022. URL: https://ctan.org/pkg/markdown (visited on 06/26/2022).

[2] The LATEX Team. *expl3. Wrapper package for experimental LATEX3.* June 16, 2022. URL: https://ctan.org/pkg/expl3 (visited on 06/26/2022).

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.