The comment2tex package

Erik Nijenhuis

2026/06/08 v1.0

Abstract

comment2tex typesets a source file that carries its documentation in special comments: the comments become ordinary LaTeX, the rest becomes a listing. It ships a Lua converter and two TeX wrappers — one for LaTeX/LuaLaTeX and one for plain TeX — both providing `\includebash` and `\includelua`.

Contents

1 Introduction

A source file may document itself: lines beginning with a chosen *doc-comment* prefix are prose, everything else is code. For Bash the prefix is a double hash (`##`); for Lua it is a triple dash (`---`). The converter `comment2tex.lua` turns such a file into LaTeX: doc lines are emitted verbatim (minus the prefix) and runs of code are wrapped in a listing.

This package is the TeX front end. `\includebash` and `\includelua` take a source file, run it through the converter, and input the result, so a document can pull in annotated sources without a manual build step.

2 Usage

`\includebash` `\includelua`

`\includebash{`⟨*file*⟩`}` typesets a Bash source (doc prefix `##`); `\includelua{`⟨*file*⟩`}` typesets a Lua source (doc prefix `---`). Both derive an output name by replacing the file's extension with `.c2t.tex`, generate that file with the appropriate *style* and *wrapper*, and `\input` it. The ⟨*file*⟩ must carry a single extension.

The LaTeX wrapper uses the `lstlisting` wrapper, so load and configure listings (this package does that for you with a plain default `\lstset`). The plain TeX wrapper uses a built-in verbatim environment instead, since listings is LaTeX-only.

3 Engines and use cases

How the conversion happens depends on the engine.

LuaLaTeX (recommended). LuaTeX embeds Lua, so the wrapper loads `comment2tex.lua` as a library and converts *in process* through `\directlua`. No shell escape, no separate run: just `lualatex document`.

pdfLaTeX with shell escape. pdfTeX cannot run Lua, so with `--shell-escape` the wrapper calls `texlua comment2tex.lua` through `\write18` to generate the fragment, then inputs it. Run `pdflatex -shell-escape document`.

pdfLaTeX, separate run (no shell escape). Generate the fragments ahead of time and let a normal run merely `\input` them. For each source run `texlua comment2tex.lua --style` ⟨*style*⟩ `-o` ⟨*base*⟩`.c2t.tex` ⟨*file*⟩ (or `make`), then `pdflatex document`. If a required fragment is missing the package raises an error telling you which command to run.

plain TeX. `comment2tex.tex` provides the same two macros for plain TeX. Under `luatex` it converts in process; under `pdftex`/`tex` it uses the same shell-escape or separate-run fallback.

Engine	In process	Otherwise
LuaLaTeX	yes (`\directlua`)	—
pdfLaTeX	no	`-shell-escape` or separate run
plain `luatex`	yes	—
plain `pdftex`/`tex`	no	`-shell-escape` or separate run

In every case the generated file is ⟨*base*⟩`.c2t.tex`, so the in-process, shell-escape and separate-run routes are interchangeable.

4 The converter, by example

Because the converter is itself a literate Lua file, the cleanest demonstration is to let it document itself. The remainder of this section is produced by `\includelua{`⟨*comment2tex.lua*⟩`}` — the generated documentation demonstrates the converter's functionality under LuaLaTeX.

```
1  #!/usr/bin/env texlua
```

4.1 The converter

This file is the engine behind `\includebash` and `\includelua`. It plays two roles from one source: a command-line program run under `texlua`, and a Lua library loaded *in process* by the TEX wrappers under LuaTEX.

Run as a program it reads a source file and emits LATEX on stdout (or to `-o`). Loaded as a library — `require"comment2tex"` or `loadfile(...)("comment2tex")` — it returns a module table whose `write` entry converts a file directly, so LuaLATEX needs neither shell escape nor a separate run.

The two modes are told apart by the first vararg of the chunk: `require` (and our explicit loader) pass the module name `"comment2tex"`, whereas `texlua` passes the command-line arguments. Per the LuaTEX manual, `texlua` is a stand-alone Lua 5.3 interpreter that fills the global `arg` table just like stock `lua`.

```
local LIBRARY_MODE = (... == "comment2tex")

local M = {}
```

4.1.1 Styles and wrappers

A *style* pairs a doc-comment prefix with a listing language; a *wrapper* is the pair of `begin`/`end` lines emitted around a code block. Both are presets so that a single `--style`/`--wrapper` selects sensible defaults, while `--comment`, `--language`, `--begin` and `--end` still override any individual field.

```
M.styles = {
  bash = { comment = "##", language = "bash" },
  lua = { comment = "---", language = "{[5.3]Lua}" },
}
```

The `lstlisting` wrapper targets LATEX (the `listings` package); the `plain` wrapper targets plain TEX, where `\ctxlisting` reads its body verbatim up to a line equal to `\endctxlisting`. Templates substitute `@LANG@` with the language and `@CONT@` with `firstnumber=last,` on every block after the first (empty on the first) so numbering stays continuous.

```
M.wrappers = {
  lstlisting = {
    begin = "\\begin{lstlisting}[language=@LANG@,@CONT@numbers=left]",
    finish = "\\end{lstlisting}",
  },
  plain = {
    begin = "\\ctxlisting%",
    finish = "\\endctxlisting",
  },
}

M.defaults = {
  style = "bash",
  wrapper = "lstlisting",
  comment = nil,
  language = nil,
  begin = nil,
  finish = nil,
}
```

4.1.2 Option handling

`new_opts` layers explicit overrides on top of the defaults; `resolve` then fills any still-empty field from the chosen style and wrapper presets.

```
31 function M.new_opts(over)
32   local o = {}
33   for k, v in pairs(M.defaults) do o[k] = v end
34   if over then
35     for k, v in pairs(over) do
36       if v ~= nil then o[k] = v end
37     end
38   end
39   return o
40 end
41
42 function M.resolve(o)
43   local style = M.styles[o.style]
44   if not style then
45     error("comment2tex: unknown style: " .. tostring(o.style) .. " (
         expected bash or lua)")
46   end
47   local wrapper = M.wrappers[o.wrapper]
48   if not wrapper then
49     error("comment2tex: unknown wrapper: " .. tostring(o.wrapper) .. " (
         expected lstlisting or plain)")
50   end
51   o.comment = o.comment or style.comment
52   o.language = o.language or style.language
53   o.begin = o.begin or wrapper.begin
54   o.finish = o.finish or wrapper.finish
55   return o
56 end
```

4.1.3 Conversion

The source is walked line by line. A line that starts with the comment prefix is documentation: any open code block is closed and the line is emitted with the prefix (and one optional following space) stripped. Anything else is code: a block is opened if one is not already running and the line is emitted unchanged. A final close guarantees the listing is shut even when the file ends inside code.

```
58 function M.convert(o, lines, emit)
59   local prefix = o.comment
60   local plen = #prefix
61   local in_code = false
62   local block_count = 0
63
64   local function open_code()
65     if not in_code then
66       block_count = block_count + 1
67       local cont = block_count == 1 and "" or "firstnumber=last,"
68       local line = o.begin:gsub("@LANG@", o.language):gsub("@CONT@", cont)
69       emit(line)
```

```
70        in_code = true
71      end
72    end
73
74    local function close_code()
75      if in_code then
76        emit(o.finish)
77        in_code = false
78      end
79    end
80
81    for _, line in ipairs(lines) do
82      if line:sub(1, plen) == prefix then
83        close_code()
84        emit((line:sub(plen + 1):gsub("^ ", "")))
85      else
86        open_code()
87        emit(line)
88      end
89    end
90    close_code()
91  end
```

4.1.4 File helpers

`read_lines` slurps a file and splits it into lines, keeping a final unterminated line if present. `convert_file` returns the converted LaTeX as a string; `write_file` sends it to `outfile`.

```
93  function M.read_lines(path)
94    local fh, err = io.open(path, "r")
95    if not fh then error("comment2tex: cannot open input: " .. tostring(err))
          end
96    local data = fh:read("*a")
97    fh:close()
98    local lines = {}
99    for line in (data .. "\n"):gmatch("(.-)\n") do
100     lines[#lines + 1] = line
101   end
102   if data:sub(-1) == "\n" then
103     lines[#lines] = nil
104   end
105   return lines
106 end
107
108 function M.convert_file(path, o)
109   o = M.resolve(o or M.new_opts())
110   local out = {}
111   M.convert(o, M.read_lines(path), function(line) out[#out + 1] = line end)
112   return table.concat(out, "\n") .. "\n"
113 end
114
115 function M.write_file(infile, outfile, o)
116   local text = M.convert_file(infile, o)
```

```
117   local fh, err = io.open(outfile, "w")
118   if not fh then error("comment2tex: cannot open output: " .. tostring(err)
          ) end
119   fh:write(text)
120   fh:close()
121   return outfile
122 end
```

4.1.5 The TEX-facing entry point

`\includebash` and `\includelua` call this through `\directlua`: it converts `infile` to `outfile` for the given style and wrapper, entirely in process. Keeping the signature positional keeps the TEX side trivial.

```
124 function M.write(style, wrapper, infile, outfile)
125   return M.write_file(infile, outfile, M.new_opts{ style = style, wrapper =
          wrapper })
126 end
```

4.1.6 Command-line interface

Parsing mirrors the documented options; `die` reports to stderr and exits non-zero. Only reached when the file is executed by `texlua`, never when it is loaded as a library.

```
128 local function usage(stream)
129   stream:write([[
130 Usage: comment2tex.lua [options] <input>
131
132 Convert a source file with embedded LaTeX doc-comments to LaTeX.
133
134 Options:
135   -s, --style NAME preset: bash (##) or lua (---) [default: bash]
136   -w, --wrapper NAME listing wrapper: lstlisting or plain [default:
          lstlisting]
137   -c, --comment PREFIX doc-comment prefix marking a doc line
138   -l, --language LANG listing language for code blocks
139   -b, --begin TEMPLATE listing begin template (@LANG@, @CONT@)
140   -e, --end TEMPLATE listing end template
141   -o, --output FILE write LaTeX here instead of stdout
142   -h, --help show this help
143
144 Templates substitute @LANG@ with the language and @CONT@ with
145 "firstnumber=last," on continuation blocks (empty on the first).
146 ]])
147 end
148
149 local function die(msg)
150   io.stderr:write("comment2tex: " .. msg .. "\n")
151   os.exit(1)
152 end
153
154 function M.main(argv)
155   local over = {}
```

```
  local input
  local i = 1
  local function value(flag)
    i = i + 1
    local v = argv[i]
    if v == nil then die("missing value for " .. flag) end
    return v
  end
  while i <= #argv do
    local a = argv[i]
    if a == "-h" or a == "--help" then
      usage(io.stdout); return 0
    elseif a == "-s" or a == "--style" then
      over.style = value(a)
    elseif a == "-w" or a == "--wrapper" then
      over.wrapper = value(a)
    elseif a == "-c" or a == "--comment" then
      over.comment = value(a)
    elseif a == "-l" or a == "--language" then
      over.language = value(a)
    elseif a == "-b" or a == "--begin" then
      over.begin = value(a)
    elseif a == "-e" or a == "--end" then
      over.finish = value(a)
    elseif a == "-o" or a == "--output" then
      over.output = value(a)
    elseif a == "--" then
      input = argv[i + 1]; break
    elseif a:sub(1, 1) == "-" and a ~= "-" then
      die("unknown option: " .. a)
    elseif input == nil then
      input = a
    else
      die("unexpected argument: " .. a)
    end
    i = i + 1
  end

  if not input then
    usage(io.stderr); return 1
  end

  local ok, err = pcall(function()
    local o = M.resolve(M.new_opts(over))
    local text = M.convert_file(input, o)
    if over.output then
      M.write_file(input, over.output, o)
    else
      io.stdout:write(text)
    end
  end)
  if not ok then die(tostring(err):gsub("^comment2tex: ", "")) end
  return 0
end
```

```
210
211 if not LIBRARY_MODE then
212   os.exit(M.main(arg))
213 end
214
215 return M
```

5 Implementation

5.1 The LaTeX wrapper

Announce the package and load listings with a neutral default style.

```
1 ⟨*package⟩
2 \NeedsTeXFormat{LaTeX2e}
3 \ProvidesPackage{comment2tex}
4   [2026/06/08 v1.0 Include annotated source files as LaTeX listings]
5 \RequirePackage{listings}
6 \lstset{basicstyle=\ttfamily\small,columns=fullflexible,
7   breaklines=true,showstringspaces=false}
```

The name of the converter, overridable before loading if it is not on the current directory or TEX tree.

```
8 \providecommand\ctxscript{comment2tex.lua}
```

Under LuaTeX, load the converter once into the global `comment2tex`, locating it through `kpse`.

```
9 \ifdefined\directlua
10   \directlua{
11     if not comment2tex then
12       local f = kpse.find_file("comment2tex.lua","lua")
13         or kpse.find_file("comment2tex.lua") or "comment2tex.lua"
14       comment2tex = assert(loadfile(f))("comment2tex")
15     end
16   }
17 \fi
```

`\ctx@include` Strip the extension, build the output name, generate it the best way the engine allows, then input it. `\ctx@strip` keeps everything before the first dot.

```
18 \def\ctx@strip#1.#2\ctx@stop{#1}
19 \newcommand\ctx@include[2]{%
20   \edef\ctx@in{#2}%
21   \edef\ctx@base{\ctx@strip#2\ctx@stop}%
22   \edef\ctx@out{\ctx@base.c2t.tex}%
23   \ifdefined\directlua
24     \directlua{comment2tex.write("#1","lstlisting",
25       "\luaescapestring{\ctx@in}","\luaescapestring{\ctx@out}")}%
26   \else
27     \ifdefined\pdfshellescape
28       \ifnum\pdfshellescape>0\relax
29         \immediate\write18{texlua \ctxscript\space
30           --style #1 --wrapper lstlisting -o \ctx@out\space \ctx@in}%
31       \fi
32     \fi
```

```
  \fi
  \IfFileExists{\ctx@out}{\input{\ctx@out}}{%
    \PackageError{comment2tex}{Converted file \ctx@out\space not found}%
      {Use LuaLaTeX, run pdflatex with -shell-escape, or pre-build it:^^J%
       texlua \ctxscript\space --style #1 -o \ctx@out\space \ctx@in}}%
}
```

`\includebash` `\includelua` The two public entry points fix the style.

```
\newcommand\includebash[1]{\ctx@include{bash}{#1}}
\newcommand\includelua[1]{\ctx@include{lua}{#1}}
⟨/package⟩
```

5.2 The plain TEX wrapper

Plain TEX has neither listings nor the LATEX file helpers, so the wrapper carries its own verbatim listing and uses primitive file tests. Control-sequence names are letters only (no digits, no `@`).

Locate and load the converter under LuaTEX, exactly as the package does.

```
⟨*plain⟩
\def\ctxscript{comment2tex.lua}
\expandafter\ifx\csname directlua\endcsname\relax\else
  \directlua{
    if not comment2tex then
      local f = kpse.find_file("comment2tex.lua","lua")
        or kpse.find_file("comment2tex.lua") or "comment2tex.lua"
      comment2tex = assert(loadfile(f))("comment2tex")
    end
  }
\fi
```

`\ctxendmark` The end-of-listing sentinel, stored as a string of category-12 characters so it can be compared against a verbatim line. A temporary escape character (`|`) lets us define it while the backslash is category 12 (“other”).

```
\begingroup
\catcode`\|=0 \catcode`\\=12
|gdef|ctxendmark{\endctxlisting}%
|endgroup
```

`\ctxuncatcode` Make the usual specials (and the space) category 12, the verbatim regime.

```
\def\ctxuncatcode{\def\do##1{\catcode`##1=12 }\dospecials}
```

`\ctxlisting` `\ctxgrab` `\ctxlisting` (emitted by the converter as `\ctxlisting%`, the trailing comment swallowing its own line end) opens a group, switches to the verbatim regime with an active end-of-line, and starts grabbing lines. `\ctxgrab` collects one line up to the active `^^M` and hands it on. Every physical line in this group ends with `%` so no stray active end-of-line leaks into the definitions.

```
\begingroup
\catcode`\^^M=\active%
\gdef\ctxlisting{%
  \par\begingroup%
  \ctxuncatcode%
  \parindent=0pt \tt%
```

```
  \catcode`\^^M=\active%
  \ctxgrab}%
\gdef\ctxgrab#1^^M{\ctxcheckend{#1}}%
\endgroup
```

`\ctxcheckend` Compare the grabbed line with the sentinel: if equal, finish; otherwise typeset it and loop. `\ctxnext` defers the recursion until after `\fi`.

```
\def\ctxcheckend#1{%
  \def\ctxtmp{#1}%
  \ifx\ctxtmp\ctxendmark
    \let\ctxnext\ctxfinish
  \else
    \leavevmode#1\par
    \let\ctxnext\ctxgrab
  \fi
  \ctxnext}
\def\ctxfinish{\par\endgroup}
```

`\ctxinclude` Mirror of the LaTeX `\ctx@include`: derive the output name, generate it in process (LuaTeX) or via shell escape (pdfTeX), then input it, erroring if it is still missing.

```
\def\ctxstrip#1.#2\ctxstop{#1}
\def\ctxinclude#1#2{%
  \edef\ctxin{#2}%
  \edef\ctxbase{\ctxstrip#2\ctxstop}%
  \edef\ctxout{\ctxbase.c2t.tex}%
  \expandafter\ifx\csname directlua\endcsname\relax
    \expandafter\ifx\csname pdfshellescape\endcsname\relax\else
      \ifnum\pdfshellescape>0
        \immediate\write18{texlua \ctxscript\space
          --style #1 --wrapper plain -o \ctxout\space \ctxin}%
      \fi
    \fi
  \else
    \directlua{comment2tex.write("#1","plain",
      "\luaescapestring{\ctxin}","\luaescapestring{\ctxout}")}%
  \fi
  \openin0=\ctxout\relax
  \ifeof0
    \closein0
    \errmessage{comment2tex: \ctxout\space not found
      (use luatex, -shell-escape, or pre-run texlua)}%
  \else
    \closein0
    \input \ctxout\relax
  \fi}
```

`\includebash` `\includelua` The public entry points.

```
\def\includebash{\ctxinclude{bash}}
\def\includelua{\ctxinclude{lua}}
```

⟨/plain⟩