The luacolor package

Heiko Oberdiek*

2021-02-17 v1.17

Abstract

Package luacolor implements color support based on LuaTEX's node attributes.

Contents

---

*Please report any issues at `https://github.com/ho-tex/luacolor/issues`

1 Documentation

1.1 Introduction

This package uses a LuaTEX's attribute register to to annotate nodes with color information. If a color is set, then the attribute register is set to this color and all nodes created in its scope (current group) are annotated with this attribute. Now the color property behaves much the same way as the font property.

1.2 Usage

Package color is loaded automatically by this package luacolor. If you need a special driver option or you prefer package xcolor, then load it before package luacolor, for example:

```
\usepackage[dvipdfmx]{xcolor}
```

The package luacolor is loaded without options:

```
\usepackage{luacolor}
```

It is able to detect PDF mode and DVI drivers are differentiated by its color specials. Therefore the package do need driver options.

Then it redefines the color setting commands to set attributes instead of whatsits for color.

At last the attribute annotations of the nodes in the output box must be analyzed to insert the necessary color whatsits. Currently LuaTEX lacks an appropriate callback function. Therefore package atbegshi is used to get control before a box is shipped out.

`\luacolorProcessBox {⟨box⟩}`

Macro `\luacolorProcessBox` processes the box ⟨*box*⟩ in the previously described manner. It is automatically called for pages, but not for XForm objects. Before passing a box to `\pdfxform`, call `\luacolorProcessBox` first.

1.3 Limitations

Ligatures with different colored components: Package luacolor sees the ligature after the paragraph building and page breaking, when a page is to be shipped out. Therefore it cannot break ligatures, because the components might occupy different space. Therefore it is the respondibility of the ligature forming process to deal with different colored glyphs that form a ligature. The user can avoid the problem entirely by explicitly breaking the ligature at the places where the color changes.

...

2 Implementation

```
⟨*package⟩
```

2.1 Catcodes and identification

```
\begingroup\catcode61\catcode48\catcode32=10\relax%
  \catcode13=5 % ^^M
  \endlinechar=13 %
  \catcode123=1 % {
  \catcode125=2 % }
  \catcode64=11 % @
  \def\x{\endgroup
    \expandafter\edef\csname LuaCol@AtEnd\endcsname{%
      \endlinechar=\the\endlinechar\relax
      \catcode13=\the\catcode13\relax
      \catcode32=\the\catcode32\relax
      \catcode35=\the\catcode35\relax
      \catcode61=\the\catcode61\relax
      \catcode64=\the\catcode64\relax
      \catcode123=\the\catcode123\relax
      \catcode125=\the\catcode125\relax
    }%
  }%
\x\catcode61\catcode48\catcode32=10\relax%
\catcode13=5 % ^^M
\endlinechar=13 %
\catcode35=6 % #
\catcode64=11 % @
\catcode123=1 % {
\catcode125=2 % }
\def\TMP@EnsureCode#1#2{%
  \edef\LuaCol@AtEnd{%
    \LuaCol@AtEnd
    \catcode#1=\the\catcode#1\relax
  }%
  \catcode#1=#2\relax
}
\TMP@EnsureCode{34}{12}% "
\TMP@EnsureCode{39}{12}% '
\TMP@EnsureCode{40}{12}% (
\TMP@EnsureCode{41}{12}% )
\TMP@EnsureCode{42}{12}% *
\TMP@EnsureCode{43}{12}% +
\TMP@EnsureCode{44}{12}% ,
\TMP@EnsureCode{45}{12}% -
```

```
\TMP@EnsureCode{46}{12}% .
\TMP@EnsureCode{47}{12}% /
\TMP@EnsureCode{58}{12}% :
\TMP@EnsureCode{60}{12}% <
\TMP@EnsureCode{62}{12}% >
\TMP@EnsureCode{91}{12}% [
\TMP@EnsureCode{93}{12}% ]
\TMP@EnsureCode{95}{12}% _ (other!)
\TMP@EnsureCode{96}{12}% ‘
\edef\LuaCol@AtEnd{\LuaCol@AtEnd\noexpand\endinput}
```

Package identification.

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{luacolor}%
  [2021-02-17 v1.17 Color support via LuaTeX’s attributes (HO)]
```

2.2 Check for LuaTEX

Without LuaTEX there is no point in using this package.

```
\RequirePackage{color}
```

```
\ifx\directlua\@undefined
  \PackageError{luacolor}{%
    This package may only be run using LuaTeX%
  }\@ehc
  \expandafter\LuaCol@AtEnd
\fi%
```

2.3 Check for disabled colors

```
\ifcolors@
\else
  \PackageWarningNoLine{luacolor}{%
    Colors are disabled by option ‘monochrome’%
  }%
  \def\set@color{}%
  \def\reset@color{}%
  \def\set@page@color{}%
  \def\define@color#1#2{}%
  \expandafter\LuaCol@AtEnd
\fi%
```

2.4 Load module and check version

```
\directlua{%
  require("luacolor")%
}
```

```
\begingroup
  \edef\x{\directlua{tex.write("2021-02-17 v1.17")}}%
  \edef\y{%
    \directlua{%
      if oberdiek.luacolor.getversion then %
        oberdiek.luacolor.getversion()%
      end%
    }%
  }%
  \ifx\x\y
  \else
    \PackageError{luacolor}{%
      Wrong version of lua module.\MessageBreak
      Package version: \x\MessageBreak
      Lua module: \y
    }\@ehc
  \fi
\endgroup
```

2.5 Find driver

```
\ifnum\outputmode=\@ne
\else
  \begingroup
    \def\current@color{}%
    \def\reset@color{}%
    \setbox\z@=\hbox{%
      \begingroup
        \set@color
      \endgroup
    }%
    \edef\reserved@a{%
      \directlua{%
        oberdiek.luacolor.dvidetect()%
      }%
    }%
    \ifx\reserved@a\@empty
      \PackageError{luacolor}{%
        DVI driver detection failed because of\MessageBreak
        unrecognized color \string\special
      }\@ehc
      \endgroup
      \expandafter\expandafter\expandafter\LuaCol@AtEnd
    \else
      \PackageInfo{luacolor}{%
        Type of color \string\special: \reserved@a
      \@gobble}%
    \fi%
  \endgroup
\fi
```

2.6 Attribute setting

`\LuaCol@Attribute`

```
\newattribute\LuaCol@Attribute
\let\LuaCol@setattribute\setattribute
\directlua{%
  oberdiek.luacolor.setattribute(\number\allocationnumber)%
}
```

`\set@color`

```
\protected\def\set@color{%
  \LuaCol@setattribute\LuaCol@Attribute{%
    \directlua{%
      oberdiek.luacolor.get("\luaescapestring{\current@color}")%
    }%
  }%
}
```

`\reset@color`

```
\def\reset@color{}
```

2.7 Whatsit insertion

\luacolorProcessBox

```
\def\luacolorProcessBox#1{%
  \directlua{%
    oberdiek.luacolor.process(\number#1)%
  }%
}
```

```
\directlua{%
  if luatexbase.callbacktypes.pre_shipout_filter then
    token.get_next()
  end
}\@secondoftwo\@gobble{
  \RequirePackage{atbegshi}[2011/01/30]
  \AtBeginShipout{%
    \luacolorProcessBox\AtBeginShipoutBox
  }
}
```

Set default color.

```
\set@color
```

2.8 `\pdfxform`/`\saveboxresource` support

```
\ifnum\outputmode=\@ne
    \let\LuaCol@org@pdfxform\saveboxresource
```

First we need some helpers to allow expandable code to parse keyword style arguments:

```
    \def\LuaCol@iii@i@ii#1#2#3{#3{#1}{#2}}
    \def\LuaCol@ii@i#1#2{{#2#1}}
    \def\LuaCol@if@keyword#1#2#3{%
      \expanded{\unexpanded{\LuaCol@iii@i@ii{#2}{#3}}\expandafter}%
      \directlua{%
        token.put_next(token.create(token.scan_keyword(token.scan_string())
        and ’@firstoftwo’
        or ’@secondoftwo’))
      }{#1}%
    }
```

The following macro scans a integer and expands to a token equivalent to a chardef whose value corresponds to the scanned integer. This allows the integer to be passed around as a undelimited argument.

```
    \def\LuaCol@scan@number{%
      \directlua{
        token.put_next(token.new(token.scan_int(), token.command_id’char_given’))
      }%
    }
```

TeX primitives like `\saveboxresource` read braced arguments in a special way. Especially they expand everything until they find a left brace. To simulate this, we use Lua to expand everything else:

```
    \def\LuaCol@scan@tobrace{%
      \directlua{
        local relax, space = token.command_id’relax’, token.command_id’spacer’
        local t
        repeat
          t = token.scan_token()
        until not (t.command == relax or t.command == space)
    token.put_next(t)
  }%
}
\def\LuaCol@scan@boxresource@i#1#2{%
  \LuaCol@if@keyword{attr}{%
    \expanded{\unexpanded{\LuaCol@scan@boxresource@iI{#1#2attr}}%
      \expandafter\expandafter\expandafter}%
    \LuaCol@scan@tobrace
  }{%
    \LuaCol@scan@boxresource@ii{#1#2}%
  }%
}
\def\LuaCol@scan@boxresource@iI#1#2{\LuaCol@scan@boxresource@ii{#1{#2}}}
\def\LuaCol@scan@boxresource@ii#1{%
  \LuaCol@if@keyword{resources}{%
    \expanded{\unexpanded{\LuaCol@scan@boxresource@iiI{#1resources}}%
      \expandafter\expandafter\expandafter}%
    \LuaCol@scan@tobrace
  }{%
    \LuaCol@scan@boxresource@iii{#1}%
  }%
}
\def\LuaCol@scan@boxresource@iiI#1#2{\LuaCol@scan@boxresource@iii{#1{#2}}}
\def\LuaCol@scan@boxresource@iii#1{%
  \LuaCol@if@keyword{margin}{%
    \expanded{\unexpanded{\LuaCol@scan@boxresource@iv{#1margin }}%
      \expandafter\expandafter\expandafter}%
    \LuaCol@scan@number
  }{%
    \LuaCol@scan@boxresource@iv{#1}{}%
  }%
}
\def\LuaCol@scan@boxresource@iv#1#2{%
  \expanded{\unexpanded{\LuaCol@scan@boxresource@v{#1#2}}%
    \expandafter\expandafter\expandafter}%
  \LuaCol@scan@number
}
\def\LuaCol@scan@boxresource@v#1#2{%
  \luacolorProcessBox{#2}%
  \LuaCol@org@pdfxform#1#2%
}

```

This could be written in Lua, but at least upto LuaTEX 1.11, feeding back too many tokens from Lua to TEX triggers a segmentation fault. This is written in Lua so the integer setting is expandable and does not interfere with a preceding `\immediate`.

```
\protected\def\saveboxresource{%
  \LuaCol@if@keyword{type}{%
    \expandafter
    \expanded{\unexpanded{\LuaCol@scan@boxresource@i{type }}%
      \expandafter\expandafter\expandafter}%
    \LuaCol@scan@number
  }{%
    \LuaCol@scan@boxresource@i{}{}%
  }%
}
```

Legacy alias.

```
    \let\pdfxform\saveboxresource
\fi
\LuaCol@AtEnd%
⟨/package⟩
```

2.9 Lua module

```
⟨*lua⟩
```

Box zero contains a `\hbox` with the color `\special`. That is analyzed to get the prefix for the color setting `\special`.

```
oberdiek = oberdiek or {}
local luacolor = oberdiek.luacolor or {}
oberdiek.luacolor = luacolor
```

`getversion()`

```
function luacolor.getversion()
  tex.write("2021-02-17 v1.17")
end
```

2.9.1 Driver detection

```
local ifpdf = tonumber(tex.outputmode or tex.pdfoutput) > 0
local prefix
local prefixes = {
  dvips   = "color ",
  dvipdfm = "pdf:sc ",
  truetex = "textcolor:",
  pctexps = "ps::",
}
local patterns = {
  ["^color "]            = "dvips",
  ["^pdf: *begincolor "] = "dvipdfm",
  ["^pdf: *bcolor "]     = "dvipdfm",
  ["^pdf: *bc "]         = "dvipdfm",
  ["^pdf: *setcolor "]   = "dvipdfm",
  ["^pdf: *scolor "]     = "dvipdfm",
  ["^pdf: *sc "]         = "dvipdfm",
  ["^textcolor:"]        = "truetex",
  ["^ps::"]              = "pctexps",
}
```

`info()`

```
local function info(msg, term)
  local target = "log"
  if term then
    target = "term and log"
  end
  texio.write_nl(target, "Package luacolor info: " .. msg .. ".")
  texio.write_nl(target, "")
end
```

`dvidetect()`

```
function luacolor.dvidetect()
  local v = tex.box[0]
  assert(v.id == node.id("hlist"))
  for v in node.traverse_id(node.id("whatsit"), v.head) do
    if v and v.subtype == node.subtype("special") then
      local data = v.data
      for pattern, driver in pairs(patterns) do
```

```
        if string.find(data, pattern) then
          prefix = prefixes[driver]
          tex.write(driver)
          return
        end
      end
      info("\\special{" .. data .. "}", true)
      return
    end
  end
  info("Missing \\special", true)
end
```

2.9.2 Color strings

```
local map = {
  n = 0,
}
```

`get()`

```
function luacolor.get(color)
  tex.write("" .. luacolor.getvalue(color))
end
```

`getvalue()`

```
function luacolor.getvalue(color)
  local n = map[color]
  if not n then
    n = map.n + 1
    map.n = n
    map[n] = color
    map[color] = n
  end
  return n
end
```

2.9.3 Attribute register

`setattribute()`

```
local attribute
function luacolor.setattribute(attr)
  attribute = attr
end
```

`getattribute()`

```
function luacolor.getattribute()
  return attribute
end
```

2.9.4 Whatsit insertion

```
local LIST = 1
local LIST_LEADERS = 2
local LIST_DISC = 3
local COLOR = 4
local NOCOLOR = 5
local RULE = node.id("rule")
local node_types = {
  [node.id("hlist")] = LIST,
  [node.id("vlist")] = LIST,
  [node.id("rule")]  = COLOR,
  [node.id("glyph")] = COLOR,
  [node.id("disc")]  = LIST_DISC,
  [node.id("whatsit")] = {
    [node.subtype("pdf_colorstack")] =
      function(n)
        return n.stack == 0 and NOCOLOR or nil
      end,
    [node.subtype("special")] = COLOR,
    [node.subtype("pdf_literal")] = COLOR,
    [node.subtype("pdf_save")] = COLOR,
    [node.subtype("pdf_restore")] = COLOR, -- probably not needed
-- TODO (DPC)    [node.subtype("pdf_refximage")] = COLOR,
  },
  [node.id("glue")] =
    function(n)
      if n.subtype >= 100 then -- leaders
        if n.leader.id == RULE then
          return COLOR
        else
          return LIST_LEADERS
        end
      end
    end,
}
```

`get_type()`

```
local function get_type(n)
  local ret = node_types[n.id]
  if type(ret) == 'table' then
    ret = ret[n.subtype]
  end
  if type(ret) == 'function' then
    ret = ret(n)
  end
  return ret
end
```

```
local mode = 2 -- luatex.pdfliteral.direct
local WHATSIT = node.id("whatsit")
local SPECIAL = node.subtype("special")
local PDFLITERAL = node.subtype("pdf_literal")
local DRY_FALSE = false
local DRY_TRUE = true
```

`traverse()`

```
local function traverse(list, color, dry)
  if not list then
    return color
  end
  local head
  if get_type(list) == LIST then
    head = list.head
  elseif get_type(list) == LIST_DISC then
    head = list.replace
  else
    texio.write_nl("!!! Error: Wrong list type: " .. node.type(list.id))
    return color
  end
⟨debug⟩texio.write_nl("traverse: " .. node.type(list.id))
  for n in node.traverse(head) do
⟨debug⟩texio.write_nl("  node: " .. node.type(n.id))
    local t = get_type(n)
⟨debug⟩texio.write_nl("TYPE "..tostring(t).. " "..tostring(node.type(node.getid(n)))..” ”.. tos
    if t == LIST or t == LIST_DISC then
      color = traverse(n, color, dry)
    elseif t == LIST_LEADERS then
      local color_after = traverse(n.leader, color, DRY_TRUE)
      if color == color_after then
        traverse(n.leader, color, DRY_FALSE or dry)
      else
        traverse(n.leader, ’’, DRY_FALSE or dry)
```

The color status is unknown here, because the leader box will or will not be set.

```
        color = ’’
      end
    elseif t == COLOR then
      local v = node.has_attribute(n, attribute)
      if v then
        local newColor = map[v]
        if newColor ~= color then
          color = newColor
          if dry == DRY_FALSE then
            local newNode
            if ifpdf then
              newNode = node.new(WHATSIT, PDFLITERAL)
              newNode.mode = mode
              newNode.data = color
            else
              newNode = node.new(WHATSIT, SPECIAL)
              newNode.data = prefix .. color
            end
            head = node.insert_before(head, n, newNode)
          end
        end
      end
    elseif t == NOCOLOR then
      color = ’’
    end
  end
  if get_type(list) == LIST then
    list.head = head
  else
    list.replace = head
  end
  return color
end
```

`process()`

```
function luacolor.process(box)
  local color = ""
  local list = tex.getbox(box)
  traverse(list, color, DRY_FALSE)
end

if luatexbase.callbacktypes.pre_shipout_filter then
```

```
  luatexbase.add_to_callback("pre_shipout_filter", function(list)
    traverse(list, "", DRY_FALSE)
    return true
  end, "luacolor.process")
end
```

For recent versions of luaotfload, we can register a callback to control how coloring glyph is handled for the color feature.

```
if luaotfload.set_colorhandler then
  local set_attribute = node.direct.set_attribute
  luaotfload.set_colorhandler(function(head, n, color)
    set_attribute(n, attribute, luacolor.getvalue(color))
    return head, n
  end)
end
```

```
⟨/lua⟩
```

3 Installation

3.1 Download

Package. This package is available on CTAN[1]:

CTAN:macros/latex/contrib/luacolor/luacolor.dtx The source file.

CTAN:macros/latex/contrib/luacolor/luacolor.pdf Documentation.

Bundle. All the packages of the bundle 'luacolor' are also available in a TDS compliant ZIP archive. There the packages are already unpacked and the documentation files are generated. The files and directories obey the TDS standard.

CTAN:install/macros/latex/contrib/luacolor.tds.zip

TDS refers to the standard "A Directory Structure for TEX Files" (CTAN:pkg/tds). Directories with `texmf` in their name are usually organized this way.

3.2 Bundle installation

Unpacking. Unpack the `luacolor.tds.zip` in the TDS tree (also known as `texmf` tree) of your choice. Example (linux):

```
unzip luacolor.tds.zip -d ~/texmf
```

Script installation. Check the directory `TDS:scripts/luacolor/` for scripts that need further installation steps.

3.3 Package installation

Unpacking. The `.dtx` file is a self-extracting `docstrip` archive. The files are extracted by running the `.dtx` through plain TEX:

```
tex luacolor.dtx
```

---

[1] CTAN:pkg/luacolor

TDS. Now the different files must be moved into the different directories in your installation TDS tree (also known as `texmf` tree):

`luacolor.sty` → `tex/latex/luacolor/luacolor.sty`
`luacolor.lua` → `scripts/luacolor/luacolor.lua`
`luacolor.pdf` → `doc/latex/luacolor/luacolor.pdf`
`luacolor.dtx` → `source/latex/luacolor/luacolor.dtx`

If you have a `docstrip.cfg` that configures and enables docstrip's TDS installing feature, then some files can already be in the right place, see the documentation of docstrip.

3.4 Refresh file name databases

If your TeX distribution (TeX Live, MiKTeX, ...) relies on file name databases, you must refresh these. For example, TeX Live users run `texhash` or `mktexlsr`.

3.5 Some details for the interested

Unpacking with LaTeX. The `.dtx` chooses its action depending on the format:

plain TeX: Run docstrip and extract the files.

LaTeX: Generate the documentation.

If you insist on using LaTeX for docstrip (really, docstrip does not need LaTeX), then inform the autodetect routine about your intention:

```
latex \let\install=y\input{luacolor.dtx}
```

Do not forget to quote the argument according to the demands of your shell.

Generating the documentation. You can use both the `.dtx` or the `.drv` to generate the documentation. The process can be configured by the configuration file `ltxdoc.cfg`. For instance, put this line into this file, if you want to have A4 as paper format:

```
\PassOptionsToClass{a4paper}{article}
```

An example follows how to generate the documentation with pdfLaTeX:

```
pdflatex luacolor.dtx
makeindex -s gind.ist luacolor.idx
pdflatex luacolor.dtx
makeindex -s gind.ist luacolor.idx
pdflatex luacolor.dtx
```

4 History

[2007/12/12 v1.0]

- First public version.

[2009/04/10 v1.1]

- Fixes for changed syntax of `\directlua` in LuaTeX 0.36.

[2010/03/09 v1.2]

- Adaptation for package luatex 2010/03/09 v0.4.

[2010/12/13 v1.3]

- Support for `\pdfxform` added.
- Loaded package luatexbase-attr recognized.
- Update for LuaTEX: 'list' fields renamed to 'head' in v0.65.0.

[2011/03/29 v1.4]

- Avoid whatsit insertion if option monochrome is used (thanks Manuel Pégourié-Gonnard).

[2011/04/22 v1.5]

- Bug fix by Manuel Pégourié-Gonnard: A typo prevented the detection of whatsits and applying color changes for `\pdfliteral` and `\special` nodes that might contain typesetting material.
- Bug fix by Manuel Pégourié-Gonnard: Now colors are also applied to leader boxes.
- Unnecessary color settings are removed for leaders boxes, if after the leader box the color has not changed. The costs are a little runtime, leader boxes are processed twice.
- Additional whatsits that are colored: `pdf_refximage`.
- Workaround for bug with `node.insert_before` removed for the version after LuaTEX 0.65, because bug was fixed in 0.27. (Thanks Manuel Pégourié-Gonnard.)

[2011/04/23 v1.6]

- Bug fix for nested leader boxes.
- Bug fix for leader boxes that change color, but are not set because of missing place.
- Version check for Lua module added.

[2011/10/22 v1.7]

- Lua functions `getattribute` and `getvalue` added to tell other external Lua functions the attribute register number for coloring.

[2011/11/01 v1.8]

- Use of `node.subtype` instead of magic numbers.

[2016/05/13 v1.9]

- More use of `node.subtype` instead of magic numbers.
- luatex 85 updates

[2016/05/16 v1.10]

- Documentation updates.

[2018/11/22 v1.11]

- handle issue 43.
- removed pre-0.65 stuff

[2019/07/25 v1.12]

- removed uses of module function, see PR70

[2019/11/29 v1.13]

- Documentation updates.
- Use iftex directly.

[2020-02-22 v1.14]

- Drop use of iftex ltxcmds and infwarerr.
- Assume ltluatex preloaded into format (true since 2015).
- Patch `\saveboxresource` rather than `\pdfxform` (keep old name as alias).
- Grab the number via Lua so that a `\immediate` prefix still works with `\saveboxresource`/`\pdfxform`.
- Added handler for the color feature of luaotfload

[2020-02-24 v1.15]

- Grab all possible arguments for `\saveboxresource`/`\pdfxform`

[2020-04-04 v1.16]

- Reset color after `pdf_colorstack` whatsits.

[2021-02-17 v1.17]

- Use $\LaTeX\,2_\varepsilon$'s new `pre_shipout_filter` callback if it's available to allow coloring background and foregrund layer material

5 Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; plain numbers refer to the code lines where the entry is used.