The Lua-UL package[*]

Marcel Krüger

tex@2krueger.de

May 31, 2022

1 User-level interface

Lua-UL uses new capabilities of the LuaTeX engine to provide underlining/ strikethrough/highlighting etc. support without breaking ligatures, kerning or restricting input. The predefined user-level commands are `\underLine`, `\highLight`, and `\strikeThrough`. (`\highLight` will only work correctly if the `luacolor` package is loaded) They are used as

```
\documentclass{article}
\usepackage{lua-ul}
\begin{document}
This package is \strikeThrough{useless}\underLine{awesome}!
\end{document}
```

This package is ~~useless~~awesome!

For limited compatibility with `soul`, the `soul` package option allows you to use the traditional macro names from `soul` instead:

```
\documentclass{article}
\usepackage[soul]{lua-ul}
\begin{document}
This package is \st{useless}\ul{awesome}!
\end{document}
```

The `\highLight` command highlights the argument in yellow by default. This color can be changed either by providing a color as optional argument or by changing the default through `\LuaULSetHighLightColor`:

---

[*]This document corresponds to Lua-UL v0.2.0, dated 2022-05-31.

```
\documentclass{article}
\usepackage{xcolor,luacolor,lua-ul}
\LuaULSetHighLightColor{green}
\begin{document}
Lots of stuff is \highLight{important enough to be highlighted},
but only few things are dangerous enough to deserve
\highLight[red]{red highlighting.}

\LuaULSetHighLightColor{yellow}
Let's go back to traditional \highLight{highlighting}.
\end{document}
```

> Lots of stuff is ==important enough to be highlighted==, but only few things are dangerous enough to deserve ==red highlighting.==
> Let's go back to traditional ==highlighting==.

The `\underLine` allows an optional argument accepting keyval options to customize the underline. Currently four keys are implemented:

textcolor Active by default. The underline is colored in the current text color.

color Allows setting the color of the underline according to the syntax the xcolor manual documents for PSTricks.

top/bottom/height At most two of these three options should be set. Specifies the height of the upper/lower edge of the underline or the linewidth.

2   Expert interface

`\newunderlinetype` Sometimes, you might try to solve more interesting problems than boring underlining, strikethrough or highlighting. Maybe you always wanted to be your own spell checker, want to demonstrate your love for ducks or you think that the traditional `\strikeThrough` is not visible enough. For all these cases, you can define your own "underlining" command based on a TeX `\[cxg]leaders` command.

For this, use

`\newunderlinetype`⟨*macro name*⟩`[`⟨*context specifier*⟩`]{`⟨`\leaders` *command*⟩`}`

First, you have to pass the name of the command which should enable your new kind of underlining. (If you want to have an additional command which takes an argument and underlines this argument, you will have to define this manually.) The optional argument provides a "context": This "context" has to expand to a string (something that can appear in a csname) which changes if the leaders box should be recalculated. The leader will be cached and reused whenever the context evaluates to the same string. So if your leaders should depend on the fontsize, the expansion of the context should contain the font

size. If you leaders contain text in the current font, your context should include `\fontname`. The default context includes the current size of `1ex` and the current color if luacolor is loaded.

The final argument contains the actual leader command. You should omit the final glue normally passed to `\leaders`, so e.g. write `\leaders\hbox{ . }` without appending `\hfill` or `\hskip1pt` etc. In most cases, the height and depth of your underlines is passed on to TeX to ensure that a deep underline does not intercept other lines. On the other hand, running dimensions work fine if you use a rule.

For example, the special underline commands demonstrated above are implemented as

```
\usepackage{luacolor,tikzducks,pict2e}
\newunderlinetype\beginUnderDuck{\cleaders\hbox{%
  \begin{tikzpicture}[x=.5ex,y=.5ex,baseline=.8ex]%
    \duck
  \end{tikzpicture}%
}}
\NewDocumentCommand\underDuck{+m}{{\beginUnderDuck#1}}
\newunderlinetype\beginUnderWavy[\number\dimexpr1ex]{\cleaders\hbox{%
    \setlength\unitlength{.3ex}%
    \begin{picture}(4,0)(0,1)
      \thicklines
      \color{red}%
      \qbezier(0,0)(1,1)(2,0)
      \qbezier(2,0)(3,-1)(4,0)
    \end{picture}%
}}
\NewDocumentCommand\underWavy{+m}{{\beginUnderWavy#1}}
\newunderlinetype\beginStrikeThough{\leaders\hbox{%
    \normalfont\bfseries/%
}}
\NewDocumentCommand\StrikeThough{+m}{{\beginStrikeThough#1}}
```

Here `\underWavy` uses a custom context because it doesn't change depending on the current font color.

If you only want to use `\newunderlinetype` and do not want to use the predefined underline types, you can use the `minimal` package option to disable them.

3   The implementation

3.1   Helper modules

First we need a separate Lua module `pre_append_to_vlist_filter` which provides a variant of the `append_to_vlist_filter` callback which can be used

by multiple packages. This ensures that we are compatible with other packages implementing `append_to_vlist_filter`. First check if an equivalent to `pre_append_to_vlist_filter` already exists. The idea is that this might eventually get added to the kernel directly.

```
if luatexbase.callbacktypes.pre_append_to_vlist_filter then
  return
end

local call_callback = luatexbase.call_callback
local flush_node = node.flush_node
local prepend_prevdepth = node.prepend_prevdepth
local callback_define
```

HACK: Do not do this at home! We need to define the engine callback directly, so we use the debug library to get the "real" `callback.define`:

```
for i=1,5 do
local name, func = require'debug'.getupvalue(luatexbase.disable_callback, i)
  if name == 'callback_register' then
    callback_define = func
    break
  end
end
if not callback_define then
  error[[Unable to find callback.define]]
end

local function filtered_append_to_vlist_filter(box,
                                                locationcode,
                                                prevdepth,
                                                mirrored)
  local current = call_callback("pre_append_to_vlist_filter",
                          box, locationcode, prevdepth,
                          mirrored)
  if not current then
    flush_node(box)
    return
  elseif current == true then
    current = box
  end
  return call_callback("append_to_vlist_filter",
                    current, locationcode, prevdepth, mirrored)
end

callback_define('append_to_vlist_filter',
              filtered_append_to_vlist_filter)
luatexbase.callbacktypes.append_to_vlist_filter = nil
luatexbase.create_callback('append_to_vlist_filter', 'exclusive',
                        function(n, _, prevdepth)
                            return prepend_prevdepth(n, prevdepth)
```

```
                          end)
    luatexbase.create_callback('pre_append_to_vlist_filter',
                          'list', false)
```

Additionally we have a module `lua-ul-patches-preserve-attr` for patches of external code to make it more compatible with attribute usage.

Currently this only caontains redefinitions of kernel commands. First some local definitions

```
local getfont = font.getfont

local direct = node.direct

local getattr = direct.getattributelist
local getid = direct.getid
local getpenalty = direct.getpenalty
local getprev = direct.getprev
local getwidth = direct.getwidth

local setattr = direct.setattributelist
local setkern = direct.setkern

local insert_after = direct.insert_after
local is_glyph = direct.is_glyph
local newnode = direct.new
local todirect = direct.todirect
local tonode = direct.tonode

local glue_id = node.id'glue'
local kern_t = node.id'kern'
local penalty_id = node.id'penalty'

local italcorr_sub
for i, n in next, node.subtypes'kern' do
  if n == 'italiccorrection' then italcorr_sub = i break end
end
assert(italcorr_sub)

local nests = tex.nest
```

Now we come to the interesting part: We redefine `\sw@slant` from the LaTeX kernel. The original definition uses `\unskip` and `\unpenalty` to remove glue and penalties and then inserts italic correction before them. Then it inserts the removed penalty and skip again. This looses the right subtype, attributes and properties of the removed nodes, so we insert the italic correction kern directly at the right position instead. When the character does not exists we still add a 0pt italic correction to stay as compatible as possible with the `\/` primitive used in the original implementation.

```
local funcid = luatexbase.new_luafunction'sw@slant'
token.set_lua('sw@slant', funcid, 'protected')
```

```
lua.get_functions_table()[funcid] = function()
  local nest = nests.top
  local tail, after = todirect(nest.tail), nil
  local id = getid(tail)
  if id == glue_id then
    if getwidth(tail) == 0 then return end
    tail, after = getprev(tail), tail
    id = getid(tail)
  end
  if id == penalty_id then
    if getpenalty(tail) == 0 then return end
    tail, after = getprev(tail), tail
  end
  local cid, fontid = is_glyph(tail)
  if not cid then return end
  local fontdir = getfont(fontid)
  local characters = fontdir and fontdir.characters
  local char = characters and characters[cid]
  local kern = newnode(kern_t, italcorr_sub)
  setkern(kern, char and char.italic or 0)
  setattr(kern, getattr(tail))
% We lie about the head and ignore the return value since tail is never nil
  insert_after(tail, tail, kern)
  if not after then nest.tail = tonode(kern) end
end
```

3.2   Lua module

Now we can define our main Lua module:

```
local unset_t = node.id'unset'
local hlist_t = node.id'hlist'
local vlist_t = node.id'vlist'
local kern_t = node.id'kern'
local glue_t = node.id'glue'

local properties = node.direct.get_properties_table()
```

`current_attr` is not `.direct` since it's used in place of a node callback argument.

```
local current_attr = node.current_attr
local has_attribute = node.direct.has_attribute
local set_attribute = node.direct.set_attribute
local dimensions = node.direct.dimensions
local flush_node = node.direct.flush_node
local getattr = node.direct.getattributelist
local getboth = node.direct.getboth
local getfield = node.direct.getfield
local getglue = node.direct.getglue
local getleader = node.direct.getleader
```

6

```
local getlist = node.direct.getlist
local setheight = node.direct.setheight
local setdepth = node.direct.setdepth
local getheight = node.direct.getheight
local getdepth = node.direct.getdepth
local getnext = node.direct.getnext
local getshift = node.direct.getshift
local insert_after = node.direct.insert_after
local insert_before = node.direct.insert_before
local nodecopy = node.direct.copy
local nodenew = node.direct.new
local setboth = node.direct.setboth
local setlink = node.direct.setlink
local hpack = node.direct.hpack
local setfield = node.direct.setfield
local slide = node.direct.slide
local setattr = node.direct.setattributelist
local setglue = node.direct.setglue
local setnext = node.direct.setnext
local setshift = node.direct.setshift
local todirect = node.direct.todirect
local tonode = node.direct.tonode
local traverse = node.direct.traverse
local traverse_id = node.direct.traverse_id
local traverse_list = node.direct.traverse_list
% The following two are needed to deal with unset nodes
local getList = function(n) return getfield(n, 'list') end
local setList = function(n, h) return setfield(n, 'list', h) end

local tokennew = token.new
local set_lua = token.set_lua
local scan_keyword = token.scan_keyword
local scan_list = token.scan_list
local scan_int = token.scan_int
local scan_toks = token.scan_toks
local put_next = token.put_next
local texerror = tex.error

local functions = lua.get_functions_table()
local char_given = token.command_id'char_given'

local underlineattrs = {}
local underline_types = {}
local underline_strict_flag = {}
local underline_over_flag = {}

local vmode do
  for k, v in pairs(tex.getmodevalues()) do
  if v == "vertical" then
      vmode = k
```

```
        break
      end
    end
  end
  local texnest = tex.nest
```

To avoid influence from \everyhbox we reset \everyhbox to an empty token
list directly before scanning. As an added complication, we need to use a name
which is guaranteed to be the primitive tokenlist and we might have to restore it
before reading the actual argument (There might be a reason why \everyhbox
was sat after all. Also we have to ensure that braces swallowed by LuaTeX are
balanced, otherwise we get hard to trace errors in alignment contexts.

```
local scan_raw_hlist do
  local create = token.create
  local lbrace, rbrace = token.new(0x7B, 1), token.new(0x7D, 2)
  tex.enableprimitives('luaul@', {'everyhbox'})
  local set_everyhbox do
    local set_toks1, set_toks2 = {create'immediateassignment',
                                  create'luaul@everyhbox', lbrace},
                                 {rbrace, create'relax'}
    function set_everyhbox(t)
      token.put_next(set_toks2)
      token.put_next(t)
      token.put_next(set_toks1)
      token.scan_token()
    end
  end
  local func = luatexbase.new_luafunction"luaul.restore_everyhbox"
  local everyhbox_saved
  functions[func] = function() set_everyhbox(everyhbox_saved) end
  local toks = {rbrace, -- Housekeeping, only for balance reasons
                lbrace, create'the', create'luaul@everyhbox', rbrace,
                create'hpack', lbrace,
                  token.new(func, token.command_id'lua_call')}
  function scan_raw_hlist()
    assert(token.get_next().command == 1)
    put_next(toks)
    token.get_next() -- Scan a corresponding brace to keep TeX's brace tracking happy
    local saved_toks = scan_toks(false, true)
    everyhbox_saved = saved_toks
    set_everyhbox{}
    local list = scan_list()
    set_everyhbox(saved_toks)
    return list
  end
end

local saved_values = {}
local function new_underline_type()
```

```
    for i=1,#underlineattrs do
      local attr = underlineattrs[i]
      saved_values[i] = tex.attribute[attr]
      tex.attribute[attr] = -0x7FFFFFFF
    end
    local strict_flag = scan_keyword'strict'
    local over_flag = scan_keyword'over'
    local b = todirect(scan_raw_hlist())
    for i=1,#underlineattrs do
      tex.attribute[underlineattrs[i]] = saved_values[i]
    end
    local lead = getlist(b)
    if not getleader(lead) then
      texerror("Leader required", {"An underline type has to \z
        be defined by leader. You should use one of the", "commands \z
        \\leaders, \\cleaders, or \\xleader, or \\gleaders here."})
    else
      local after = getnext(lead)
      if after then
        texerror("Too many nodes", {"An underline type can only be \z
            defined by a single leaders specification,", "not by \z
            multiple nodes. Maybe you supplied an additional glue?",
            "Anyway, the additional nodes will be ignored"})
        setnext(lead, nil)
      end
      table.insert(underline_types, lead)
      setList(b, after)
      flush_node(b)
    end
    put_next(tokennew(#underline_types, char_given))
    underline_strict_flag[#underline_types] = strict_flag
    underline_over_flag[#underline_types] = over_flag
  end
```

In `append_to_vlist_filter` we can not access the list attributes, so we just take the current ones. They might be incorrect if the attribute changes in the vlist, so we record the original value in a property then.

```
  local function set_underline()
    local j, props
    for i=texnest.ptr,0,-1 do
      local mode = texnest[i].mode
      if mode == vmode or mode == -vmode then
        local head = todirect(texnest[i].head)
        local head_props = properties[head]
        if not head_props then
          head_props = {}
          properties[head] = head_props
        end
        props = head_props.luaul_attributes
        if not props then
```

9

```
          props = {}
          head_props.luaul_attributes = props
          break
        end
      end
    end
    for i=1,#underlineattrs do
      local attr = underlineattrs[i]
      if tex.attribute[attr] == -0x7FFFFFFF then
        j = attr
        break
      end
    end
    if not j then
      j = luatexbase.new_attribute(
          "luaul" .. tostring(#underlineattrs+1))
      underlineattrs[#underlineattrs+1] = j
    end
    props[j] = props[j] or -0x7FFFFFFF
    tex.attribute[j] = scan_int()
  end

  local function reset_underline()
    local reset_all = scan_keyword'*'
    local j
    for i=1,#underlineattrs do
      local attr = underlineattrs[i]
      if tex.attribute[attr] ~= -0x7FFFFFFF then
        if reset_all then
          tex.attribute[attr] = -0x7FFFFFFF
        else
          j = attr
        end
      end
    end
    if not j then
      if not reset_all then
        texerror("No underline active", {"You tried to disable \z
                underlining but underlining was not active in the first",
                "place. Maybe you wanted to ensure that \z
                no underling can be active anymore?", "Then you should \z
                append a *."})
      end
      return
    end
    tex.attribute[j] = -0x7FFFFFFF
  end
  local new_underline_type_func =
      luatexbase.new_luafunction"luaul.new_underline_type"
  local set_underline_func =
```

```
    luatexbase.new_luafunction"luaul.set_underline_func"
local reset_underline_func =
    luatexbase.new_luafunction"luaul.reset_underline_func"
set_lua("LuaULNewUnderlineType", new_underline_type_func)
set_lua("LuaULSetUnderline", set_underline_func, "protected")
set_lua("LuaULResetUnderline", reset_underline_func, "protected")
functions[new_underline_type_func] = new_underline_type
functions[set_underline_func] = set_underline
functions[reset_underline_func] = reset_underline
```

A little helper to measure box contents and creating a glue node with inverted dimensions.

```
local stretch_fi = {}
local shrink_fi = {}
local function fil_levels(n)
  for i=0,4 do
    stretch_fi[i], shrink_fi[i] = 0, 0
  end
  for n in traverse_id(glue_t, n) do
    local w, st, sh, sto, sho = getglue(n)
    stretch_fi[sto] = stretch_fi[sto] + st
    shrink_fi[sho] = shrink_fi[sho] + sh
  end
  local stretch, shrink = 0, 0
  for i=0,4 do
    if stretch_fi[i] ~= 0 then
      stretch = i
    end
    if shrink_fi[i] ~= 0 then
      shrink = i
    end
  end
  return stretch, shrink
end
local function new_glue_neg_dimensions(n, t,
                                       stretch_order, shrink_order)
  local g = nodenew(glue_t)
  local w = -dimensions(n, t)
  setglue(g, w)
%  setglue(g, -dimensions(n, t), 0, 0, stretch_order, shrink_order)
  setnext(g, n)
  setglue(g, w, -dimensions(1, 1, stretch_order, g, t),
                dimensions(1, 2, shrink_order, g, t),
                stretch_order, shrink_order)
  setnext(g, nil)
  return g
end
```

Now the actual underlining

```
local add_underline_hlist, add_underline_hbox, add_underline_vbox
local function add_underline_vlist(head, attr, outervalue)
  local iter, state, n = traverse_list(head) -- FIXME: unset nodes
  local t
  n, t = iter(state, n)
  while n ~= nil do
    local real_new_value = has_attribute(n, attr)
    local new_value = real_new_value ~= outervalue
                        and real_new_value or nil
    if underline_strict_flag[new_value] or not new_value then
      if t == hlist_t then
        add_underline_hbox(n, attr, real_new_value)
      elseif t == vlist_t then
        add_underline_vbox(n, attr, real_new_value)
      end
      n, t = iter(state, n)
    elseif real_new_value <= 0 then
      n, t = iter(state, n)
    else
      local nn
      nn, t = iter(state, n)
      local prev, next = getboth(n)
      setboth(n, nil, nil)
      local shift = getshift(n)
      setshift(n, 0)
      local new_list = hpack((add_underline_hlist(n, attr)))
      setheight(new_list, getheight(n))
      setdepth(new_list, getdepth(n))
      setshift(new_list, shift)
      setlink(prev, new_list, next)
      set_attribute(new_list, attr, 0)
      if n == head then
        head = new_list
      end
      n = nn
    end
  end
  return head
end
function add_underline_vbox(head, attr, outervalue)
  if outervalue and outervalue <= 0 then return end
  setList(head, add_underline_vlist(getList(head), attr, outervalue))
  set_attribute(head, attr, outervalue and -outervalue or 0)
end
function add_underline_hlist(head, attr, outervalue)
  local max_height, max_depth
  slide(head)
  local last_value
```

```lua
local first
local shrink_order, stretch_order
for n, id, subtype in traverse(head) do
  local real_new_value = has_attribute(n, attr)
  local new_value
  if real_new_value then
    if real_new_value > 0 then
      set_attribute(n, attr, -real_new_value)
      new_value = real_new_value ~= outervalue
                   and real_new_value or nil
    end
  else
    set_attribute(n, attr, 0)
  end
  if id == hlist_t then
    if underline_strict_flag[new_value]
        or subtype == 3 or not new_value then
      add_underline_hbox(n, attr, real_new_value)
      new_value = nil
    end
  elseif id == vlist_t then
    if underline_strict_flag[new_value] or not new_value then
      add_underline_vbox(n, attr, real_new_value)
      new_value = nil
    end
  elseif id == kern_t and subtype == 0 then
    local after = getnext(n)
    if after then
      local next_value = has_attribute(after, attr)
      if next_value == outervalue or not next_value then
        new_value = nil
      else
        new_value = last_value
      end
    else
      new_value = last_value
    end
  elseif id == glue_t and (
      subtype == 8 or
      subtype == 9 or
      subtype == 15 or
  false) then
    new_value = nil
  end
  if last_value ~= new_value then
    if not stretch_order then
      stretch_order, shrink_order = fil_levels(head)
    end
    if last_value then
```

If the value changed and the old one wasn't `nil`, then we reached the end of the previous underlined segment and therefore know it's length. Therefore we can finally insert the underline.

Currently both the underline and the corresponding negative glue inherit the attributes from when the underline was defined. This makes sure that these nodes get consistent attributes (avoiding e.g. that only one of the nodes being picked up in a later pass and therefore interfering with the underlining) and that these are as much as possible under use control.

We can't really predict what the most sensible value for attributes we don't control is, but by using this way any issues should be fixable with by adjusting the context argument.

Currently this block gets duplicated a few lines down for the end of the list. This should get refactored into it's own function, but I have to be careful to handle all the special cases there.

```
          local glue = new_glue_neg_dimensions(first, n,
              stretch_order, shrink_order)
          local w, st, sh = getglue(glue)
          local lead = nodecopy(underline_types[last_value])
          setglue(lead, -w, -st, -sh, stretch_order, shrink_order)
          setattr(glue, getattr(lead))
          if underline_over_flag[last_value] then
            head = insert_before(head, n, glue)
            insert_after(head, glue, lead)
          else
            head = insert_before(head, first, lead)
            insert_after(head, lead, glue)
          end
        end
        if new_value then
          first = n
          local box = getleader(underline_types[new_value])
          if not max_height or getheight(box) > max_height then
            max_height = getheight(box)
          end
          if not max_depth or getdepth(box) > max_depth then
            max_depth = getdepth(box)
          end
        end
        last_value = new_value
      end
    end
    if last_value then
      local glue = new_glue_neg_dimensions(first, nil,
          stretch_order, shrink_order)
      local w, st, sh = getglue(glue)
      local lead = nodecopy(underline_types[last_value])
      setglue(lead, -w, -st, -sh, stretch_order, shrink_order)
      setattr(glue, getattr(lead))
```

```lua
      if underline_over_flag[last_value] then
        insert_before(head, nil, glue)
        insert_after(head, glue, lead)
      else
        head = insert_before(head, first, lead)
        insert_after(head, lead, glue)
      end
    end
  end
  return head, max_height, max_depth
end
function add_underline_hbox(head, attr, outervalue, set_height_depth)
  if outervalue and outervalue <= 0 then return end
  local new_head, new_height, new_depth
      = add_underline_hlist(getList(head), attr, outervalue)
  setList(head, new_head)
  if set_height_depth then
    if new_height and getheight(head) < new_height then
      setheight(head, new_height)
    end
    if new_depth and getdepth(head) < new_depth then
      setdepth(head, new_depth)
    end
  end
  set_attribute(head, attr, outervalue and -outervalue or 0)
end
require'pre_append_to_vlist_filter'
luatexbase.add_to_callback('pre_append_to_vlist_filter',
    function(b, loc, prev, mirror)
      local props = properties[todirect(texnest.top.head)]
      props = props and props.luaul_attributes
      b = todirect(b)
      if loc == "post_linebreak" then
        for i = 1, #underlineattrs do
          local attr = underlineattrs[i]
          local current = props and props[attr] or tex.attribute[attr]
          if current == -0x7FFFFFFF then
            current = nil
          end
          add_underline_hbox(b, underlineattrs[i], current, true)
        end
      else
        for i = 1, #underlineattrs do
          local attr = underlineattrs[i]
          local current = props and props[attr] or tex.attribute[attr]
          local b_attr = has_attribute(b, attr)
          if b_attr and b_attr ~= current then
            local shift = getshift(b)
            setshift(b, 0)
            b = hpack((add_underline_hlist(b, attr)))
            setshift(b, shift)
```

```
            set_attribute(b, attr, 0)
          end
        end
      end
      return tonode(b)
    end, 'add underlines to list')
  luatexbase.add_to_callback('hpack_filter',
      function(head, group, size, pack, dir, attr)
```

When `hpack_filter` is called as part of an alignment, no attributes are passed. It seems like a bug, but we will just substitute with the current attributes. Since the callbacks are called after the group for the cell ended, these should always be right.

```
      if group == 'align_set' or group == 'fin_row' then
        attr = current_attr()
      end
      head = todirect(head)
      for i = 1, #underlineattrs do
        local ulattr = underlineattrs[i]
        local current
        for n in node.traverse(attr) do
          if n.number == ulattr then
            current = n.value
          end
        end
        head = add_underline_hlist(head, ulattr, current)
      end
      return tonode(head)
    end, 'add underlines to list')
  luatexbase.add_to_callback('vpack_filter',
      function(head, group, size, pack, maxdepth, dir, attr)
  %      if true then return head end
      head = todirect(head)
      for i = 1, #underlineattrs do
        local ulattr = underlineattrs[i]
        local current
        for n in node.traverse(attr) do
          if n.number == ulattr then
            current = n.value
          end
        end
        head = add_underline_vlist(head, ulattr, current)
      end
      return tonode(head)
    end, 'add underlines to list')
```

Finally load `lua-ul-patches-preserve-attr`.

```
  require'lua-ul-patches-preserve-attr'
```

3.3 TₑX support package

Now only some LaTeX glue code is still needed First define some messages.

```
\msg_new:nnnn {lua-ul} {luatex-required} {LuaLaTeX~required} {
  Lua-UL~requires~LuaLaTeX.~
  Maybe~you~forgot~to~switch~the~engine~in~your~editor?
}
\msg_new:nnnn {lua-ul} {luacolor-required} {
  Color~support~in~lua-ul~requires~luacolor.
}{
  This~error~can~be~fixed~by~adding~\token_to_str:N \usepackage{luacolor}~in~your~preamble.
}
```

Only LuaLaTeX is supported. For other engines we show an error.

```
\sys_if_engine_luatex:F {
  \msg_error:nn {lua-ul} {luatex-required}
}
\lua_load_module:n {lua-ul}
%
```

We support some options. Especially `minimal` will disable the predefined commands `\underLine` and `\strikeThrough` and allow you to define similar commands with your custom settings instead, `soul` tries to replicate names of the `soul` package.

```
\bool_new:N \l__luaul_predefined_bool
\bool_new:N \l__luaul_soulnames_bool
\bool_set_true:N \l__luaul_predefined_bool
\DeclareOption {minimal} { \bool_set_false:N \l__luaul_predefined_bool }
\DeclareOption {soul} { \bool_set_true:N \l__luaul_soulnames_bool }
\ProcessOptions\relax
```

Just one more tiny helper.

```
\cs_new_protected:Npn \__luaul_maybe_define_use:Nx #1#2 {
  \cs_if_free:NT #1 {
    \cs_gset:Npx #1 {#2}
  }
  #1
}
\cs_generate_variant:Nn \__luaul_maybe_define_use:Nx {c}
```

The default for the context argument. Give that most stuff should scale vertically with the font size, we expect most arguments to be given in `ex`. Additionally especially traditional underlines will use the currently active text color, so especially when luacolor is loaded we have to include the color attribute too.

```
\cs_new:Npn \__luaul_default_context: {
  \dim_to_decimal_in_sp:n {1ex}
  @
  \cs_if_exist:NT \LuaCol@Attribute {
    \int_use:N \LuaCol@Attribute
  }
}
```

The main macro.

```
\NewDocumentCommand \newunderlinetype
    { E{*}{{}} m O{\__luaul_default_context:} m } {
  \cs_new_protected:Npn #2 {
    \__luaul_maybe_define_use:cx {
      __luaul_set_underline_ \cs_to_str:N #2 _ #3 :
    } {
      \LuaULSetUnderline
        \LuaULNewUnderlineType #1 { #4 \hskip 0pt }
    }
  }
}


\bool_if:NT \l__luaul_predefined_bool {
  \msg_new:nnnn {lua-ul} {height-inconsistent} {Specified~height~is~inconsistent~with~specified~top-
  \msg_new:nnnn {lua-ul} {deprecated} {Used~deprecated~command~`\token_to_str:N #1'.~Please~replace-
  \cs_new:Npn \__luaul_deprecated:NN #1#2 {
    \cs_new_protected:Npn #1 {
      \msg_warning:nnnn {lua-ul} {deprecated} {#1} {#2}
      \cs_gset_eq:NN #1 #2
      #1
    }
  }
}
```

For `\highLight`, the color should be customizable. There are two cases: If xcolor is not loaded, we just accept a simple color name. Otherwise, we accept color as documented in xcolor for PSTricks: Either a color name, a color expression or a combination of colormodel and associated values.

```
\tl_new:N \l__luaul_current_color_tl
\cs_new:Npn \__luaul_set_color_strip:w \xcolor@ #1#2 {}
\cs_new_protected:Npn \luaul_set_color:n #1 {
  \cs_if_free:NTF \XC@getcolor {
    \tl_set:Nn \l__luaul_current_color_tl {#1}
  } {
    \group_begin:
      \XC@getcolor {#1} \l_tmpa_tl
    \use:e { \group_end:
      \tl_set:Nn \exp_not:N \l__luaul_current_color_tl {
        \exp_after:wN \__luaul_set_color_strip:w \l_tmpa_tl
      }
    }
  }
}
\__luaul_deprecated:NN \luaul@setcolor \luaul_set_color:n
\cs_new_protected:Npn \luaul_apply_color: {
  \IfPackageLoadedTF {luacolor} {
    \cs_if_free:NTF \XC@getcolor {
      \color {\l__luaul_current_color_tl}
    }{
      \exp_after:wN \XC@undeclaredcolor \l__luaul_current_color_tl
```

18

```
    }
  }{
    \msg_error:nn {lua-ul} {luacolor-required}
  }
}
\__luaul_deprecated:NN \luaul@applycolor \luaul_apply_color:
\cs_new:Npn \luaul_context_color: {
  \l__luaul_current_color_tl
}
\cs_new:Npn \__luaul_default_colored_context: {
  \dim_to_decimal_in_sp:n {1ex}
  @
  \luaul_context_color:
}
```

Now a user-level command to set the default color for highlights.

```
\tl_new:N \l__luaul_highlight_color_tl
\tl_set:Nn \l__luaul_highlight_color_tl {yellow}
\NewDocumentCommand \LuaULSetHighLightColor {om} {
  \tl_set:Nx \l__luaul_highlight_color_tl {
    \IfValueTF {#1} {
      [#1]{#2}
    }{
      #2
    }
  }
}
```

The sizes for the predefined commands are stolen from the "soul" default values.

```
\dim_new:N \l__luaul_top_dim
\dim_new:N \l__luaul_bottom_dim
\dim_new:N \l__luaul_height_dim
\bool_new:N \l__luaul_top_set_bool
\bool_new:N \l__luaul_bottom_set_bool
\bool_new:N \l__luaul_height_set_bool
\bool_new:N \l__luaul_color_set_bool

\keys_define:nn {luaul / generic} {
  top .code:n = {
    \bool_set_true:N \l__luaul_top_set_bool
    \dim_set:Nn \l__luaul_top_dim {#1}
  },
  bottom .code:n = {
    \bool_set_true:N \l__luaul_bottom_set_bool
    \dim_set:Nn \l__luaul_bottom_dim {(#1)*-1}
  },
  height .code:n = {
    \bool_set_true:N \l__luaul_height_set_bool
    \dim_set:Nn \l__luaul_height_dim {#1}
  },
```

```
    color .code:n = {
      \bool_set_true:N \l__luaul_color_set_bool
      \luaul_set_color:n {#1}
    },
    textcolor .code:n = {
      \bool_set_false:N \l__luaul_color_set_bool
    },
    textcolor .value_forbidden:n = true,
}

\cs_new:Npn \__luaul_parse_generic_keys:n #1 {
  \bool_set_false:N \l__luaul_top_set_bool
  \bool_set_false:N \l__luaul_bottom_set_bool
  \bool_set_false:N \l__luaul_height_set_bool

  \keys_set:nn {luaul / generic} {#1}

  \bool_if:NTF \l__luaul_bottom_set_bool {
    \bool_if:NTF \l__luaul_top_set_bool {
      \bool_if:NTF \l__luaul_height_set_bool {
        \dim_compare:nNnF { \l__luaul_top_dim + \l__luaul_bottom_dim } = {\l__luaul_height_dim} {
          \msg_error:nn {lua-ul} {height-inconsistent}
        }
      } {
        \dim_set:Nn \l__luaul_height_dim { \l__luaul_top_dim + \l__luaul_bottom_dim }
      }
    } {
      \dim_set:Nn \l__luaul_top_dim { \l__luaul_height_dim - \l__luaul_bottom_dim }
    }
  } {
    \dim_set:Nn \l__luaul_bottom_dim { \l__luaul_height_dim - \l__luaul_top_dim }
  }
}

\newunderlinetype \@@underLine [
  \dim_to_decimal_in_sp:n { \l__luaul_top_dim }
  @
  \dim_to_decimal_in_sp:n { \l__luaul_bottom_dim }
  @
  \bool_if:NTF \l__luaul_color_set_bool {
    colored @ \luaul_context_color:
  } {
    \cs_if_exist:NT \LuaCol@Attribute {
      \int_use:N \LuaCol@Attribute
    }
  }
] {
  \bool_if:NT \l__luaul_color_set_bool {
    \luaul_apply_color:
  }
```

```
    \leaders\vrule height \l__luaul_top_dim depth \l__luaul_bottom_dim
}
\newunderlinetype \@basicUnderLine {
  \leaders \vrule height -.65ex depth .75ex
}

\NewDocumentCommand \@underLine {o} {
  \IfValueTF {#1} {
    \__luaul_parse_generic_keys:n {
      top = -.65ex,
      bottom = -.75ex,
    }
    \__luaul_parse_generic_keys:n {
      #1
    }
    \@@underLine
  } {
    \@basicUnderLine
  }
}
\NewDocumentCommand \underLine {o +m} {
  \group_begin:
  \@underLine [#1] #2
  \group_end:
}

\newunderlinetype \@@strikeThrough {
  \leaders \vrule height .55ex depth -.45ex
}
\newunderlinetype \@colored@strikeThrough [\__luaul_default_colored_context:] {
  \luaul_apply_color:
  \leaders\vrule height .55ex depth -.45ex
}
\NewDocumentCommand \@strikeThrough {o} {
  \IfValueTF {#1} {
    \luaul_set_color:n {#1}
    \@colored@strikeThrough
  }
  \@@strikeThrough
}
\NewDocumentCommand \strikeThrough {o +m} {
  \group_begin:
  \@strikeThrough #2
  \group_end:
}

\newunderlinetype \@@highLight [\__luaul_default_colored_context:] {
  \luaul_apply_color:
  \leaders\vrule height 1.75ex depth .75ex
}
```

```
\NewDocumentCommand \@highLight {O{\l__luaul_highlight_color_tl}} {
  \luaul_set_color:n {#1}
  \@@highLight
}
\NewDocumentCommand \highLight {O{\l__luaul_highlight_color_tl} +m} {
  \group_begin:
  \@highLight [#1] #2
  \group_end:
}
\bool_if:NT \l__luaul_soulnames_bool {
  \cs_new_eq:NN \textul \underLine     \cs_new_eq:NN \ul \textul
  \cs_new_eq:NN \textst \strikeThrough \cs_new_eq:NN \st \textst
  \cs_new_eq:NN \texthl \highLight     \cs_new_eq:NN \hl \texthl
}
}
```

Finally patch `\reset@font` to ensure that underlines do not propagate into unexpected places.

```
\ifx \reset@font \normalfont
  \let \reset@font \relax
  \DeclareRobustCommand \reset@font {%
    \normalfont
    \LuaULResetUnderline*%
  }
\else
  \MakeRobust \reset@font
  \begingroup
    \expandafter \let
        \expandafter \helper
        \csname reset@font \endcsname
  \expandafter \endgroup
  \expandafter \gdef
    \csname reset@font \expandafter \endcsname
  \expandafter {%
    \helper%
    \LuaULResetUnderline*%
  }
\fi
```

In the output routine, the page box is repacked before `\reset@font` is called, so we have to ensure to reset the attributes before that. This will use some awesome output routine hook as soon as that's in the kernel, until then manual patching it is.

At the time I am writing this the remaining code of the package contains exactly ten times `\expandafter`. Interestingly, that's also exactly the number of `\expandafter`s we use here.

```
\output\expandafter\expandafter\expandafter{%
  \expandafter\expandafter\expandafter\LuaULResetUnderline
  \expandafter\expandafter\expandafter*%
  \expandafter\@firstofone\the\output%
```

```
}
```

Change History