1 User-level interface

Lua-UL uses new capabilities of the LuaTeX engine to provide underlining/strikelthrough/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 uselessawesome!

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}

2 Expert interface

Sometimes, you might try to solve more interesting problems than boring un-
derlining, 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 \texttt{\leaders} command.

For this, use
\begin{verbatim}
\newunderlinetype{macro name}[(context specifier)]{\leaders command}
\end{verbatim}

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 \texttt{\fontname}. The default context includes the current size of \texttt{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 \texttt{\hfill} or \texttt{\hskip1pt} etc. The height and depth of your leaders is ignored to ensure that adding underlines etc. does not change the general text layout. It is your responsibility to ensure that you are not too high or too low and 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
\begin{verbatim}
\usepackage{luacolor,tikzducks,pict2e}
\newunderlinetype\beginUnderDuck\leaders\hbox{\begin{tikzpicture}[x=.5ex,y=.5ex,baseline=.8ex]%\duck\end{tikzpicture} %}
\end{verbatim}
\begin{verbatim}
\newcommand\underDuck[1]{{\beginUnderDuck#1}}
\end{verbatim}
\begin{verbatim}
\newunderlinetype\beginUnderWavy\leaders\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} %}
\end{verbatim}
\begin{verbatim}
\newcommand\underWavy[1]{{\beginUnderWavy#1}}
\end{verbatim}
\begin{verbatim}
\newunderlinetype\beginStrikeThough\leaders\hbox{\normalfont\bfseries/\hfill}
\end{verbatim}

\end{verbatim}
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.

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

```lua
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`:

```lua
for i=1,5 do
    local name, func = 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
```

```lua
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",
    box, 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)

3.2 Lua module

Now we can define our main Lua module:

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 char_given = token.command_id'char_given'
local underlineattrs = {}
local underline_types = {}
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 b = token.scan_list()
    for i=1,#underlineattrs do
        tex.attribute[underlineattrs[i]] = saved_values[i]
    end
    local lead = b.head
    if not lead.leader then
        tex.error("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."})
    end

else
  if lead.next then
    tex.error("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")
  end
  table.insert(underline_types, lead)
  b.head = lead.next
  node.flush_node(b)
end
token.put_next(token.new(#underline_types, char_given))
end
local function set_underline()
  local j
  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
  tex.attribute[j] = token.scan_int()
end
local functions = lua.get_functions_table()
local new_underline_type_func =
  luatexbase.new_luafunction"luaul.new_underline_type"
local set_underline_func =
  luatexbase.new_luafunction"luaul.set_underline_func"
token.set_lua("LuaULNewUnderlineType", new_underline_type_func)
token.set_lua("LuaULSetUnderline", set_underline_func, "protected")
functions[new_underline_type_func] = new_underline_type
functions[set_underline_func] = set_underline
local add_underline_h
local function add_underline_v(head, attr)
  for n in node.traverse(head) do
    if head.id == hlist_t then
      add_underline_h(n, attr)
    elseif head.id == vlist_t then
      add_underline_v(n.head, attr)
    end
  end
end
function add_underline_h(head, attr)
local used = false
node.slide(head.head)
local last_value
local first
for n in node.traverse(head.head) do
  local new_value = node.has_attribute(n, attr)
  if n.id == hlist_t then
    new_value = nil
    add_underline_h(n, attr)
  elseif n.id == vlist_t then
    new_value = nil
    add_underline_v(n.head, attr)
  elseif n.id == kern_t and n.subtype == 0 then
    if n.next and not node.has_attribute(n.next, attr) then
      new_value = nil
    else
      new_value = last_value
    end
  elseif n.id == glue_t and (n.subtype == 8 or n.subtype == 9 or n.subtype == 15 or false) then
    new_value = nil
  end
  if last_value ~= new_value then
    if last_value then
      local width = node.rangedimensions(head, first, n)
      local kern = node.new(kern_t)
      kern.kern = -width
      local lead = node.copy(underline_types[last_value])
      lead.width = width
      head.head = node.insert_before(head.head, first, lead)
      node.insert_after(head, lead, kern)
    end
    if new_value then
      first = n
    end
    last_value = new_value
  end
end
if last_value then
  local width = node.rangedimensions(head, first)
  local kern = node.new(kern_t)
  kern.kern = -width
  kern.next = node.copy(underline_types[last_value])
  kern.next.width = width
  node.tail(head.head).next = kern
end
local function filter(b, loc, prev, mirror)
for i = 1, #underlineattrs do
  add_underline_v(b, underlineattrs[i])
end
return true
end
require 'pre_append_to_vlist_filter'
luatexbase.add_to_callback('pre_append_to_vlist_filter',
filter, 'add underlines to list')

3.3 TeX support package

Now only some \LaTeX glue code is still needed. Only \LaTeX\TeX is supported. For other engines we show an error.

\ifx\directlua\undefined
  \PackageError{lua-ul}{LuaLaTeX required}{}
  \{Lua-UL requires LuaLaTeX.
  Maybe you forgot to switch the engine in your editor?\}
\fi
\directlua{require 'lua-ul'}
\RequirePackage{xparse}

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

\newif\ifluaul@predefined
\newif\ifluaul@soulnames
\luaul@predefinedtrue
\DeclareOption{minimal}{\luaul@predefinedfalse}
\DeclareOption{soul}{\luaul@soulnamestrue}
\ProcessOptions{relax}

Just one more tiny helper.

\protected\def\luaul@maybedefineuse#1#2{%
  \unless\ifcsname#1\endcsname
    \expandafter\xdef\csname#1\endcsname{#2}%
  \fi
  \csname#1\endcsname
}

The main macro.

\NewDocumentCommand{\newunderlinetype}{mO{\luaul@defaultcontext}m}{%\newcommand{\#1}{} % “Reserve” the name
  \protected\def{\#1}{%
    \expandafter\luaul@maybedefineuse
    \expanded{\csstring{\#1@@\#2}}%
    \LuaULSetUnderline
    \luaULNewUnderlineType\hbox{\#3\hskip0pt}%
}