

The `luatexbase-attr` package

Heiko Oberdiek (primary author of `luatex`)
Élie Roux, Manuel Pégourié-Gonnard, Philipp Gesang*

<https://github.com/lualatex/luatexbase>
lualatex-dev@tug.org

v0.6 2013-05-11

Abstract

In addition to the registers existing in \TeX and $\epsilon\text{-}\text{\TeX}$, $\text{Lua}\text{\TeX}$ introduces a new concept: attributes. This package takes care of attribute allocation just like Plain \TeX and \LaTeX do for other registers, and also provides a Lua interface.

Contents

1	Documentation	1
1.1	\TeX interface	1
1.2	Lua interface	2
2	Implementation	2
2.1	\TeX package	2
2.1.1	Preliminaries	2
2.1.2	Primitives needed	4
2.1.3	Load supporting Lua module	4
2.2	User macros	4
2.3	Lua module	5
3	Test files	9

1 Documentation

1.1 \TeX interface

The main macro defined here is `\newluatexattribute`. It behaves in the same way as `\newcount`. There are also two helper macros: `\setluatexattribute` sets an attribute's value (locally, but you can use `\global` in front of it). `\unsetluatexattribute` unsets an attribute by giving it a special value, depending on $\text{Lua}\text{\TeX}$'s version; you should always use this macro in order to be sure the correct special value for your version of $\text{Lua}\text{\TeX}$ is used.

Due to the intended use of attributes, it makes no sense to locally allocate an attribute the way you can locally allocate a counter using `etex.sty`'s `\loccount`, so no corresponding macro is defined.

*See "History" in [luatexbase.pdf](#) for details.

1.2 Lua interface

The various Lua functions for manipulating attributes use a number to designate the attribute. Hence, package writers need a way to know the number of the attribute associated to `\fooattr` assuming it was defined using `\newluatexattribute\fooattr`, something that LuaTeX currently doesn't support (you can get the current value of the associated attribute as `tex.attribute.fooattr`, but not the attribute number).

There are several ways to work around this. For example, it is possible to extract the number at any time from the `\meaning` of `\fooattr`. Alternatively, one could look at `\the\allocationnumber` just after the definition of `\fooattr` and remember it in a Lua variable. For your convenience, this is automatically done by `\newluatexattribute`: the number is remembered in a dedicated Lua table so that you can get it as `luatexbase.attributes.fooattr` (mind the absence of backslash here) at any time.

Also, two Lua functions are provided that are analogous to the above TeX macros (actually, the macros are wrappers around the functions): `luatexbase.new_attribute(<name>)` allocates a new attribute, without defining a corresponding TeX control sequence (only an entry in `luatexbase.attributes` is created). It usually returns the number of the allocated attribute. If room is missing, it raises an error, unless the second argument (optional) is not false, in which case it returns -1.

`luatexbase.unset_attribute(<name>)` unsets an existing attribute.

2 Implementation

2.1 TeX package

1 (*texpackage)

2.1.1 Preliminaries

Catcode defenses and reload protection.

```
2 \begingroup\catcode61\catcode48\catcode32=10\relax% = and space
3 \catcode123 1 % {
4 \catcode125 2 % }
5 \catcode 35 6 % #
6 \toks0\expandafter{\expandafter\endlinechar\the\endlinechar}%
7 \edef\x{\endlinechar13}%
8 \def\y#1 #2 {%
9   \toks0\expandafter{\the\toks0 \catcode#1 \the\catcode#1}%
10  \edef\x{x \catcode#1 #2}}%
11 \y 13 5 % carriage return
12 \y 61 12 % =
13 \y 32 10 % space
14 \y 123 1 % {
15 \y 125 2 % }
16 \y 35 6 % #
17 \y 64 11 % @ (letter)
18 \y 10 12 % new line ^^J
19 \y 34 12 % "
20 \y 39 12 % '
21 \y 40 12 % (
22 \y 41 12 % )
23 \y 44 12 % ,
```

```

24 \y 45 12 % -
25 \y 46 12 % .
26 \y 47 12 % /
27 \y 58 12 % :
28 \y 60 12 % <
29 \y 62 12 % >
30 \y 91 12 % [
31 \y 93 12 % ]
32 \y 94 7 % ^
33 \y 95 8 % _
34 \y 96 12 % '
35 \toks0\expandafter{\the\toks0 \relax\noexpand\endinput}%
36 \edef\y#1{\noexpand\expandafter\endgroup%
37 \noexpand\ifx#1\relax \edef#1{\the\toks0}\x\relax%
38 \noexpand\else \noexpand\expandafter\noexpand\endinput%
39 \noexpand\fi}%
40 \expandafter\y\cename luatexbase@attr@sty@endinput\endcsname%

```

Package declaration.

```

41 \begingroup
42 \expandafter\ifx\cename ProvidesPackage\endcsname\relax
43 \def\x#1[#2]{\immediate\write16{Package: #1 #2}}
44 \else
45 \let\x\ProvidesPackage
46 \fi
47 \expandafter\endgroup
48 \x{luatexbase-attr}[2013/05/11 v0.6 Attributes allocation for LuaTeX]

```

Make sure LuaTeX is used.

```

49 \begingroup\expandafter\expandafter\expandafter\endgroup
50 \expandafter\ifx\cename RequirePackage\endcsname\relax
51 \input ifluatex.sty
52 \else
53 \RequirePackage{ifluatex}
54 \fi
55 \ifluatex\else
56 \begingroup
57 \expandafter\ifx\cename PackageError\endcsname\relax
58 \def\x#1#2#3{\begingroup \newlinechar10
59 \errhelp{#3}\errmessage{Package #1 error: #2}\endgroup}
60 \else
61 \let\x\PackageError
62 \fi
63 \expandafter\endgroup
64 \x{luatexbase-attr}{LuaTeX is required for this package. Aborting.}{%
65 This package can only be used with the LuaTeX engine^^J%
66 (command 'lualatex' or 'luatex').^^J%
67 Package loading has been stopped to prevent additional errors.}
68 \expandafter\luatexbase@attr@sty@endinput%
69 \fi

```

2.1.2 Primitives needed

First load `luatexbase-modutils` (hence `luatexbase-loader` and `luatexbase-compat`), and make sure `luatex.sty` is loaded too.

```
70 \begingroup\expandafter\expandafter\expandafter\endgroup
71 \expandafter\ifx\csname RequirePackage\endcsname\relax
72   \input luatexbase-modutils.sty
73   \input luatex.sty
74 \else
75   \RequirePackage{luatexbase-modutils}
76   \RequirePackage{luatex}
77 \fi
```

Make sure the primitives we need are available.

```
78 \luatexbase@ensure@primitive{luaescapestring}
79 \luatexbase@ensure@primitive{attributedef}
80 \luatexbase@ensure@primitive{attribute}
```

2.1.3 Load supporting Lua module

```
81 \luatexbase@directlua{require('luatexbase.attr')}
```

2.2 User macros

The allocation macro is merely a wrapper around the Lua function, but handles error and logging in \TeX , for consistency with other allocation macros.

```
82 \def\newluatexattribute#1{%
83   \begingroup\escapechar\m@ne \expandafter\expandafter\expandafter
84   \endgroup \expandafter\expandafter\expandafter
85   \allocationnumber \luatexbase@directlua{tex.write(
86     luatexbase.new_attribute("\luatexluaescapestring{\string#1}", true))}%
87   \ifnum\allocationnumber>\m@ne
88     \global\luatexattributedef#1=\allocationnumber
89     \wlog{\string#1=\string\luatexattribute\the\allocationnumber}%
90   \else
91     \errmessage{No room for a new \string\attribute}%
92   \fi}
```

Helper macro `\unsetluatexattribute`.

```
93 \newcount\lltxb@attr@unsetvalue
94 \lltxb@attr@unsetvalue=\ifnum\luatexversion<37 -1\else -2147483647\fi\relax
95 \def\unsetluatexattribute#1{%
96   #1\lltxb@attr@unsetvalue}
```

And now the trivial helper macro.

```
97 \def\setluatexattribute#1#2{%
98   #1=\numexpr#2\relax}
```

That's all folks!

```
99 \luatexbase@attr@sty@endinput%
100 \endpackage
```

2.3 Lua module

```

101 (*luamodule)
102 --- locals
103 local copynode      = node.copy
104 local newnode       = node.new
105 local nodesubtype   = node.subtype
106 local nodetype      = node.id
107 local stringfind    = string.find
108 local stringformat   = string.format
109 local tableunpack    = unpack or table.unpack
110 local texiowrite_nl = texio.write_nl
111 local texiowrite     = texio.write
112 --- luatex internal types
113 local whatsit_t      = nodetype"whatsit"
114 local user_defined_t = nodesubtype"user_defined"
115 local unassociated   = "__unassociated"
116 luatexbase           = luatexbase or { }
117 local luatexbase     = luatexbase
118 local err, warning, info, log = luatexbase.provides_module({
119     name      = "luatexbase-attr",
120     version   = 0.6,
121     date      = "2013/05/11",
122     description = "Attributes allocation for LuaTeX",
123     author    = "Elie Roux, Manuel Pegourie-Gonnard and Philipp Gesang",
124     copyright  = "Elie Roux, Manuel Pegourie-Gonnard and Philipp Gesang",
125     license    = "CC0",
126 })

```

This table holds the values of the allocated attributes, indexed by name.

```

127 luatexbase.attributes = luatexbase.attributes or { }
128 local attributes     = luatexbase.attributes

```

Scoping: we use locals for the attribute functions.

```

129 local new_attribute
130 local unset_attribute

```

In the LuaTeX ecosystem there are currently two functions that create a new attribute. One is in `oberdiek` bundle, the other is this one. We will hack a little in order to make them compatible. The other function uses `LuT@AllocAttribute` as attribute counter, we will keep it in sync with ours. A possible problem might also appear: the other function starts attribute allocation at 0, which will break `luaotfload`. We output an error if a new attribute has already been allocated with number 0.

```

131 local luatex_sty_counter = 'LuT@AllocAttribute'
132 if tex.count[luatex_sty_counter] then
133   if tex.count[luatex_sty_counter] > -1 then
134     error("luatexbase error: attribute 0 has already been set by \newattribute"
135         .."macro from luatex.sty, not belonging to this package, this makes"
136         .."luaotfload unusable. Please report to the maintainer of luatex.sty")
137   else
138     tex.count[luatex_sty_counter] = 0
139   end
140 end

```

The allocation function. Unlike other registers, allocate starting from 1. Some code (e. g., font handling coming from `ConTeXt`) behaves strangely with `\attribute0` set, and since there

is plenty of room here, it doesn't seem bad to "lose" one item in order to avoid this problem.

```

141 local last_alloc = 0
142 function new_attribute(name, silent)
143   if last_alloc >= 65535 then
144     if silent then
145       return -1
146     else
147       error("No room for a new \\attribute", 1)
148     end
149   end
150   local lsc = tex.count[luatex_sty_counter]
151   if lsc and lsc > last_alloc then
152     last_alloc = lsc
153   end
154   last_alloc = last_alloc + 1
155   if lsc then
156     tex.setcount('global', luatex_sty_counter, last_alloc)
157   end
158   attributes[name] = last_alloc
159   unset_attribute(name)
160   if not silent then
161     log('luatexbase.attributes[%q] = %d', name, last_alloc)
162   end
163   return last_alloc
164 end
165 luatexbase.new_attribute = new_attribute

```

Unset an attribute the correct way depending on LuaTeX's version.

```

166 local unset_value = (luatexbase.luatexversion < 37) and -1 or -2147483647
167 function unset_attribute(name)
168   tex.setattribute(attributes[name], unset_value)
169 end
170 luatexbase.unset_attribute = unset_attribute

```

User whatsit allocation (experimental).

```

171 --- cf. luatexref-t.pdf, sect. 8.1.4.25
172 local user_whatsits = { --- (package, (name, id hash)) hash
173   __unassociated = { }, --- those without package name
174 }
175 local whatsit_ids = { } --- (id, (name * package)) hash
176 local whatsit_cap = 2^53 --- Lua numbers are doubles
177 local current_whatsit = 0
178 local anonymous_whatsits = 0
179 local anonymous_prefix = "anon"

```

The whatsit allocation is split into two functions: `new_user_whatsit_id` registers a new id (an integer) and returns it. It is up to the user what he actually does with the return value.

Registering whatsits without a name, though supported, is not exactly good style. In these cases we generate a name from a counter.

In addition to the whatsit name, it is possible and even encouraged to specify the name of the package that will be using the whatsit as the second argument.

```

180 --- string -> string -> int
181 local new_user_whatsit_id = function (name, package)
182   if name then
183     if not package then

```

```

184         package = unassociated
185     end
186 else -- anonymous
187     anonymous_whatsits = anonymous_whatsits + 1
188     warning("defining anonymous user whatsit no. %d", anonymous_whatsits)
189     warning("dear package authors, please name your whatsits!")
190     package = unassociated
191     name     = anonymous_prefix .. tostring(anonymous_whatsits)
192 end
193
194 local whatsitdata = user_whatsits[package]
195 if not whatsitdata then
196     whatsitdata = { }
197     user_whatsits[package] = whatsitdata
198 end
199
200 local id = whatsitdata[name]
201 if id then --- warning
202     warning("replacing whatsit %s:%s (%d)", package, name, id)
203 else --- new id
204     current_whatsit = current_whatsit + 1
205     if current_whatsit >= whatsit_cap then
206         warning("maximum of %d integral user whatsit ids reached",
207             whatsit_cap)
208         warning("further whatsit allocation may be inconsistent")
209     end
210     id = current_whatsit
211     whatsitdata[name] = id
212     whatsit_ids[id] = { name, package }
213 end
214 log("new user-defined whatsit %d (%s:%s)", id, package, name)
215 return id
216 end
217 luatexbase.new_user_whatsit_id = new_user_whatsit_id

    new_user_whatsit first registers a new id and then also creates the corresponding whatsit
    of subtype "user defined". We return a nullary function that delivers copies of the whatsit.
218 --- string -> string -> (unit -> node_t, int)
219 local new_user_whatsit = function (name, package)
220     local id = new_user_whatsit_id(name, package)
221     local whatsit = newnode(whatsit_t, user_defined_t)
222     whatsit.user_id = id
223     --- unit -> node_t
224     return function ( ) return copynode(whatsit) end, id
225 end
226 luatexbase.new_user_whatsit = new_user_whatsit
227 luatexbase.new_user_whatsit_factory = new_user_whatsit --- for Stephan

    If one knows the name of a whatsit, its corresponding id can be retrieved by means of
    get_user_whatsit_id.
228 --- string -> string -> int
229 local get_user_whatsit_id = function (name, package)
230     if not package then
231         package = unassociated

```

```

232     end
233     return user_whatsits[package][name]
234 end
235 luatexbase.get_user_whatsit_id = get_user_whatsit_id

```

The inverse lookup is also possible via `get_user_whatsit_name`. Here it finally becomes obvious why it is beneficial to supply a package name – it adds information about who created and might be relying on the whatsit in question. First return value is the whatsit name, the second the package identifier it was registered with.

We issue a warning and return empty strings in case the asked whatsit is unregistered.

```

236 --- int | fun | node -> (string, string)
237 local get_user_whatsit_name = function (asked)
238     local id
239     if type(asked) == "number" then
240         id = asked
241     elseif type(asked) == "function" then
242         --- node generator
243         local n = asked()
244         id = n.user_id
245     else --- node
246         id = asked.user_id
247     end
248     local metadata = whatsit_ids[id]
249     if not metadata then -- unknown
250         warning("whatsit id %d unregistered; inconsistencies may arise", id)
251         return "", ""
252     end
253     return tableunpack(metadata)
254 end
255 luatexbase.get_user_whatsit_name = get_user_whatsit_name

```

For the curious as well as the cautious who are interesting in what they are dealing with, we add a function that outputs the current allocation status to the terminal.

```

256 --- string -> unit
257 local dump_registered_whatsits = function (asked_package)
258     local whatsit_list = { }
259     if asked_package then
260         local whatsitdata = user_whatsits[asked_package]
261         if not whatsitdata then
262             error("(no user whatsits registered for package %s)",
263                 asked_package)
264             return
265         end
266         texiowrite_nl("(user whatsit allocation stats for " .. asked_package)
267         for name, id in next, whatsitdata do
268             whatsit_list[#whatsit_list+1] =
269                 stringformat("(%s:%s %d)", asked_package, name, id)
270         end
271     else
272         texiowrite_nl("(user whatsit allocation stats")
273         texiowrite_nl(stringformat(" ((total %d)\n (anonymous %d))",
274             current_whatsit, anonymous_whatsits))
275         for package, whatsitdata in next, user_whatsits do
276             for name, id in next, whatsitdata do

```



```

277         whatsit_list[#whatsit_list+1] =
278             stringformat("(%s:%s %d)", package, name, id)
279     end
280 end
281 end
282
283 texiowrite_nl" ("
284 --- in an attempt to be clever the texio.write* functions
285 --- mess up line breaking, so concatenation is unusable ...
286 local first = true
287 for i=1, #whatsit_list do
288     if first then
289         first = false
290     else -- indent
291         texiowrite_nl"  "
292     end
293     texiowrite(whatsit_list[i])
294 end
295 texiowrite"))\n"
296 end
297 luatexbase.dump_registered_whatsits = dump_registered_whatsits

```

Lastly, we define a couple synonyms for convenience.

```

298 luatexbase.newattribute           = new_attribute
299 luatexbase.newuserwhatsit        = new_user_whatsit
300 luatexbase.newuserwhatsitfactory = new_user_whatsit_factory
301 luatexbase.newuserwhatsitid       = new_user_whatsit_id
302 luatexbase.getuserwhatsitid       = get_user_whatsit_id
303 luatexbase.getuserwhatsitname     = get_user_whatsit_name
304 luatexbase.dumpregisteredwhatsits = dump_registered_whatsits
305 </luamodule>

```

3 Test files

The tests done are very basic: we just make sure that the package loads correctly and the macros don't generate any error, under both L^AT_EX and Plain T_EX. We also check that the attribute's number is remembered well, independently of the current value of `\escapechar`.

```

306 <testplain>\input luatexbase-attr.sty
307 <testlatex>\RequirePackage{luatexbase-attr}
308 <*testplain,testlatex>
309 \newluatexattribute\testattr
310 \setluatexattribute\testattr{1}
311 \ifnum\testattr=1 \else \ERROR \fi
312 \unsetluatexattribute\testattr
313 \ifnum\testattr<0 \else \ERROR \fi
314 \catcode64 11
315 \luatexbase@directlua{assert(luatexbase.attributes.testattr)}
316 \luatexbase@directlua{luatexbase.new_attribute('luatestattr')}
317 \luatexbase@directlua{assert(luatexbase.attributes.luatestattr)}
318 \begingroup
319 \escapechar64
320 \newluatexattribute\anotherattr
321 \endgroup

```

```
322 \setluatexattribute\anotherattr{1}  
323 \luatexbase@directlua{assert(luatexbase.attributes.anotherattr)}  
324 \testplain, testlatex}  
325 \testplain\bye  
326 \testlatex\stop
```