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 ε-TEX, LuaTEX 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 TEX interface

The main macro defined here is `\newluatexattribute`. It behaves in the same way as `\newcount`. There are also two helper macros: `\setluatexattibute` 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 LuaTEX's version; you should always use this macro in order to be sure the correct special value for your version of LuaTEX 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 % ,
```

```
  \y  45 12 % -
  \y  46 12 % .
  \y  47 12 % /
  \y  58 12 % :
  \y  60 12 % <
  \y  62 12 % >
  \y  91 12 % [
  \y  93 12 % ]
  \y  94  7 % ^
  \y  95  8 % _
  \y  96 12 % `
  \toks0\expandafter{\the\toks0 \relax\noexpand\endinput}%
  \edef\y#1{\noexpand\expandafter\endgroup%
    \noexpand\ifx#1\relax \edef#1{\the\toks0}\x\relax%
    \noexpand\else \noexpand\expandafter\noexpand\endinput%
    \noexpand\fi}%
\expandafter\y\csname luatexbase@attr@sty@endinput\endcsname%
```

Package declaration.

```
\begingroup
  \expandafter\ifx\csname ProvidesPackage\endcsname\relax
    \def\x#1[#2]{\immediate\write16{Package: #1 #2}}
  \else
    \let\x\ProvidesPackage
  \fi
\expandafter\endgroup
\x{luatexbase-attr}[2013/05/11 v0.6 Attributes allocation for LuaTeX]
```

Make sure LuaTEX is used.

```
\begingroup\expandafter\expandafter\expandafter\endgroup
\expandafter\ifx\csname RequirePackage\endcsname\relax
  \input ifluatex.sty
\else
  \RequirePackage{ifluatex}
\fi
\ifluatex\else
  \begingroup
    \expandafter\ifx\csname PackageError\endcsname\relax
      \def\x#1#2#3{\begingroup \newlinechar10
        \errhelp{#3}\errmessage{Package #1 error: #2}\endgroup}
    \else
      \let\x\PackageError
    \fi
  \expandafter\endgroup
  \x{luatexbase-attr}{LuaTeX is required for this package. Aborting.}{%
    This package can only be used with the LuaTeX engine^^J%
    (command `lualatex' or `luatex').^^J%
    Package loading has been stopped to prevent additional errors.}
  \expandafter\luatexbase@attr@sty@endinput%
\fi
```

2.1.2 Primitives needed

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

```
\begingroup\expandafter\expandafter\expandafter\endgroup
\expandafter\ifx\csname RequirePackage\endcsname\relax
  \input luatexbase-modutils.sty
  \input luatex.sty
\else
  \RequirePackage{luatexbase-modutils}
  \RequirePackage{luatex}
\fi
```

Make sure the primitives we need are available.

```
\luatexbase@ensure@primitive{luaescapestring}
\luatexbase@ensure@primitive{attributedef}
\luatexbase@ensure@primitive{attribute}
```

2.1.3 Load supporting Lua module

```
\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.

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

Helper macro `\unsetluatexattribute`.

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

And now the trivial helper macro.

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

That's all folks!

```
\luatexbase@attr@sty@endinput%
⟨/texpackage⟩
```

2.3 Lua module

```
⟨*luamodule⟩
--- locals
local copynode          = node.copy
local newnode           = node.new
local nodesubtype       = node.subtype
local nodetype          = node.id
local stringfind        = string.find
local stringformat      = string.format
local tableunpack       = unpack or table.unpack
local texiowrite_nl     = texio.write_nl
local texiowrite        = texio.write
--- luatex internal types
local whatsit_t         = nodetype"whatsit"
local user_defined_t    = nodesubtype"user_defined"
local unassociated      = "__unassociated"
luatexbase              = luatexbase or { }
local luatexbase        = luatexbase
local err, warning, info, log = luatexbase.provides_module({
    name          = "luatexbase-attr",
    version       = 0.6,
    date          = "2013/05/11",
    description   = "Attributes allocation for LuaTeX",
    author        = "Elie Roux, Manuel Pegourie-Gonnard and Philipp Gesang",
    copyright     = "Elie Roux, Manuel Pegourie-Gonnard and Philipp Gesang",
    license       = "CC0",
})
```

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

```
luatexbase.attributes   = luatexbase.attributes or { }
local attributes        = luatexbase.attributes
```

Scoping: we use locals for the attribute functions.

```
local new_attribute
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.

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

```
local last_alloc = 0
function new_attribute(name, silent)
    if last_alloc >= 65535 then
        if silent then
            return -1
        else
            error("No room for a new \\attribute", 1)
        end
    end
    local lsc = tex.count[luatex_sty_counter]
    if lsc and lsc > last_alloc then
      last_alloc = lsc
    end
    last_alloc = last_alloc + 1
    if lsc then
      tex.setcount('global', luatex_sty_counter, last_alloc)
    end
    attributes[name] = last_alloc
    unset_attribute(name)
    if not silent then
        log('luatexbase.attributes[%q] = %d', name, last_alloc)
    end
    return last_alloc
end
luatexbase.new_attribute = new_attribute
```

Unset an attribute the correct way depending on LuaTEX's version. The constant `unset_value` can be retrieved by calling `get_unset_value()` to apply to nodes.

```
local unset_value = (luatexbase.luatexversion < 37) and -1 or -2147483647
function unset_attribute(name)
    tex.setattribute(attributes[name], unset_value)
end
luatexbase.unset_attribute = unset_attribute
luatexbase.get_unset_value = function () return unset_value end
```

Allocation of user-defined whatsit nodes (experimental). User-defined whatsit nodes (or user whatsits) are ignored by the LuaTEX engine. They can thus be used to store information in node lists without doing any harm. User whatsits can be distinguished by an id that is stored in node field `user_id`.

```
--- cf. luatexref-t.pdf, sect. 8.1.4.25
local user_whatsits       = {    --- (package, (name, id hash)) hash
    __unassociated = { },        --- those without package name
}
local whatsit_ids         = { }  --- (id, (name * package)) hash
local whatsit_cap         = 2^53 --- Lua numbers are doubles
local current_whatsit     = 0
local anonymous_whatsits  = 0
local anonymous_prefix    = "anon"
```

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

```
--- string -> string -> int
local new_user_whatsit_id = function (name, package)
    if name then
        if not package then
            package = unassociated
        end
    else -- anonymous
        anonymous_whatsits = anonymous_whatsits + 1
        warning("defining anonymous user whatsit no. %d", anonymous_whatsits)
        warning("dear package authors, please name your whatsits!")
        package = unassociated
        name    = anonymous_prefix .. tostring(anonymous_whatsits)
    end

    local whatsitdata = user_whatsits[package]
    if not whatsitdata then
        whatsitdata             = { }
        user_whatsits[package]  = whatsitdata
    end

    local id = whatsitdata[name]
    if id then --- warning
        warning("replacing whatsit %s:%s (%d)", package, name, id)
    else --- new id
        current_whatsit     = current_whatsit + 1
        if current_whatsit >= whatsit_cap then
            warning("maximum of %d integral user whatsit ids reached",
                whatsit_cap)
            warning("further whatsit allocation may be inconsistent")
        end
        id                  = current_whatsit
        whatsitdata[name]   = id
        whatsit_ids[id]     = { name, package }
    end
    log("new user-defined whatsit %d (%s:%s)", id, package, name)
    return id
end
luatexbase.new_user_whatsit_id = new_user_whatsit_id
```

`new_user_whatsit` first registers a new id and then also creates the corresponding whatsit node of subtype "user-defined". We return a nullary function that delivers copies of the whatsit.

Alternatively, the first argument can be a whatsit node that will then be used as prototype. Note that in this case a *copy* of the prototype will be stored in the closure, eliminating side-effects.

```
--- (string | node_t) -> string -> ((unit -> node_t) * int)
local new_user_whatsit = function (req, package)
    local id, whatsit
    if type(req) == "string" then
        id              = new_user_whatsit_id(req, package)
        whatsit         = newnode(whatsit_t, user_defined_t)
        whatsit.user_id = id
    elseif req.id == whatsit_t and req.subtype == user_defined_t then
        id      = req.user_id
```

```
228         whatsit = copynode(req)
229         if not whatsit_ids[id] then
230             warning("whatsit id %d unregistered; "
231                     .. "inconsistencies may arise", id)
232         end
233     end
234     return function () return copynode(whatsit) end, id
235 end
236 luatexbase.new_user_whatsit         = new_user_whatsit
```

If one knows the name of a user whatsit, its corresponding id can be retrieved by means of `get_user_whatsit_id`.

```
237 --- string -> string -> int
238 local get_user_whatsit_id = function (name, package)
239     if not package then
240         package = unassociated
241     end
242     return user_whatsits[package][name]
243 end
244 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 user whatsit in question. First return value is the user whatsit name, the second the package identifier it was registered with.

We issue a warning and return empty strings in case the argument doesn't correspond to a registered user whatsit id.

```
245 --- int | fun | node -> (string, string)
246 local get_user_whatsit_name = function (asked)
247     local id
248     if type(asked) == "number" then
249         id = asked
250     elseif type(asked) == "function" then
251         --- node generator
252         local n = asked()
253         id = n.user_id
254     else --- node
255         id = asked.user_id
256     end
257     local metadata = whatsit_ids[id]
258     if not metadata then -- unknown
259         warning("whatsit id %d unregistered; inconsistencies may arise", id)
260         return "", ""
261     end
262     return tableunpack(metadata)
263 end
264 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.

```
265 --- string -> unit
266 local dump_registered_whatsits = function (asked_package)
267     local whatsit_list = { }
268     if asked_package then
```

```
        local whatsitdata = user_whatsits[asked_package]
        if not whatsitdata then
            error("(no user whatsits registered for package %s)",
                  asked_package)
            return
        end
        texiowrite_nl("(user whatsit allocation stats for " .. asked_package)
        for name, id in next, whatsitdata do
            whatsit_list[#whatsit_list+1] =
                stringformat("(%s:%s %d)", asked_package, name, id)
        end
    else
        texiowrite_nl("(user whatsit allocation stats")
        texiowrite_nl(stringformat(" ((total %d)\n  (anonymous %d))",
            current_whatsit, anonymous_whatsits))
        for package, whatsitdata in next, user_whatsits do
            for name, id in next, whatsitdata do
                whatsit_list[#whatsit_list+1] =
                    stringformat("(%s:%s %d)", package, name, id)
            end
        end
    end

    texiowrite_nl" ("
    --- in an attempt to be clever the texio.write* functions
    --- mess up line breaking, so concatenation is unusable ...
    local first = true
    for i=1, #whatsit_list do
        if first then
            first = false
        else -- indent
            texiowrite_nl"  "
        end
        texiowrite(whatsit_list[i])
    end
    texiowrite"))\n"
end
luatexbase.dump_registered_whatsits = dump_registered_whatsits
```

Lastly, we define a couple synonyms for convenience.

```
luatexbase.newattribute            = new_attribute
luatexbase.newuserwhatsit          = new_user_whatsit
luatexbase.newuserwhatsitid        = new_user_whatsit_id
luatexbase.getuserwhatsitid        = get_user_whatsit_id
luatexbase.getuserwhatsitname      = get_user_whatsit_name
luatexbase.dumpregisteredwhatsits  = dump_registered_whatsits
```

⟨/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 LaTeX and Plain TeX. We also check that the attribute's number is remembered well, independently of the current value of `\escapechar`.

```
⟨testplain⟩\input luatexbase-attr.sty
⟨testlatex⟩\RequirePackage{luatexbase-attr}
⟨*testplain, testlatex⟩
\newluatexattribute\testattr
\setluatexattribute\testattr{1}
\ifnum\testattr=1 \else \ERROR \fi
\unsetluatexattribute\testattr
\ifnum\testattr<0 \else \ERROR \fi
\catcode64 11
\luatexbase@directlua{assert(luatexbase.attributes.testattr)}
\luatexbase@directlua{luatexbase.new_attribute('luatestattr')}
\luatexbase@directlua{assert(luatexbase.attributes.luatestattr)}
\begingroup
\escapechar64
\newluatexattribute\anotherattr
\endgroup
\setluatexattribute\anotherattr{1}
\luatexbase@directlua{assert(luatexbase.attributes.anotherattr)}
⟨/testplain, testlatex⟩
⟨testplain⟩\bye
⟨testlatex⟩\stop
```