

The luatexbase-attr package

Manuel Pégourié-Gonnard & Élie Roux

Support: lualatex-dev@tug.org

v0.3 2010-05-27

Abstract

In addition to the registers existing in \TeX and $\varepsilon\text{-}\TeX$, Lua \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	3
2.1.3	Load supporting Lua module	4
2.2	User macros	4
2.3	Lua module	5
3	Test files	5

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 Lua \TeX 's version; you should always use this macro in order to be sure the correct special value for your version of Lua \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.

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 `\foobar`. 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

Reload protection, especially for Plain TeX.

```
2             \csname lltxb@attr@loaded\endcsname
3 \expandafter\let\csname lltxb@attr@loaded\endcsname\endinput

   Catcode defenses.

4 \begingroup
5 \catcode123 1 % {
6 \catcode125 2 % }
7 \catcode 35 6 % #
8 \toks0{}%
9 \def\x{}%
10 \def\y#1 #2 {%
11 \toks0\expandafter{\the\toks0 \catcode#1 \the\catcode#1}%
12 \edef\x{\x \catcode#1 #2}}%
13 \y 123 1 % {
14 \y 125 2 % }
15 \y 35 6 % #
16 \y 10 12 % ^^J
17 \y 34 12 % "
18 \y 36 3 % $ $
19 \y 39 12 % '
20 \y 40 12 % (
21 \y 41 12 % )
```

```

22 \y 42 12 % *
23 \y 43 12 % +
24 \y 44 12 % ,
25 \y 45 12 % -
26 \y 46 12 % .
27 \y 47 12 % /
28 \y 60 12 % <
29 \y 61 12 % =
30 \y 64 11 % @ (letter)
31 \y 62 12 % >
32 \y 95 12 % _ (other)
33 \y 96 12 % `
34 \edef\y#1{\endgroup\edef#1{\the\toks0\relax}\x}%
35 \expandafter\y\csname lltxb@attr@AtEnd\endcsname

Package declaration.

36 \begingroup
37 \expandafter\ifx\csname ProvidesPackage\endcsname\relax
38 \def\x#1[#2]{\immediate\write16{Package: #1 #2}}
39 \else
40 \let\x\ProvidesPackage
41 \fi
42 \expandafter\endgroup
43 \x{luatexbase-attr}[2010/10/10 v0.3 Attributes allocation for LuaTeX]

Make sure LuaTeX is used.

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

```

2.1.2 Primitives needed

Load luatexbase-compat.

```

66 \begingroup\expandafter\expandafter\expandafter\endgroup

```

```

67 \expandafter\ifx\csname RequirePackage\endcsname\relax
68   \input luatexbase-compat.sty
69 \else
70   \RequirePackage{luatexbase-compat}
71 \fi

```

Make sure the primitives we need are available.

```

72 \luatexbase@ensure@primitive{luaescapestring}
73 \luatexbase@ensure@primitive{attributedef}
74 \luatexbase@ensure@primitive{attribute}

```

2.1.3 Load supporting Lua module

First load `luatexbase-loader` (hence `luatexbase-compat`), then the supporting Lua module.

```

75 \begingroup\expandafter\expandafter\expandafter\endgroup
76 \expandafter\ifx\csname RequirePackage\endcsname\relax
77   \input luatexbase-loader.sty
78 \else
79   \RequirePackage{luatexbase-loader}
80 \fi
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`: wrapper around the Lua function.

```

93 \def\unsetluatexattribute#1{%
94   \begingroup\escapechar\m@ne
95   \luatexbase@directlua{%
96     luatexbase.unset_attribute("\luatexluaescapestring{\string#1}")}%
97   \endgroup}

```

And now the trivial helper macro.

```

98 \def\setluatexattribute#1#2{%
99   #1=\numexpr#2\relax}

```

That's all folks!

```

100 \lltxb@attr@AtEnd
101 \endpackage

```

2.3 Lua module

```
102 <luamodule>
103 module('luatexbase', package.seeall)
```

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

```
104 attributes = {}
```

The allocation function. Unlike other registers, allocate starting from 1. Some code (eg, font handling coming from ConTeXt) behaves strangely with `\attribute0` and since there is plenty of room here, it doesn't seem bad to "loose" one item in order to avoid this problem.

```
105 local last_alloc = 0
106 function new_attribute(name, silent)
107     if last_alloc >= 65535 then
108         if silent then
109             return -1
110         else
111             error("No room for a new \\attribute", 1)
112         end
113     end
114     last_alloc = last_alloc + 1
115     attributes[name] = last_alloc
116     unset_attribute(name)
117     if not silent then
118         texio.write_nl('log', string.format(
119             'luatexbase.attributes[%q] = %d', name, last_alloc))
120     end
121     return last_alloc
122 end
```

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

```
123 local unset_value = (luatexbase.luatexversion < 37) and -1 or -2147483647
124 function unset_attribute(name)
125     tex.setattribute(attributes[name], unset_value)
126 end
127 </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 en Plain TeX. We also check that the attribute's number is remembered well, independently of the current value of `\escapechar`.

```
128 <testplain>\input luatexbase-attr.sty
129 <testlatex>\RequirePackage{luatexbase-attr}
130 <*testplain, testlatex>
131 \newluatexattribute\testattr
132 \setluatexattribute\testattr{1}
133 \unsetluatexattribute\testattr
134 \catcode64 11
135 \luatexbase@directlua{assert(luatexbase.attributes.testattr)}
136 \luatexbase@directlua{luatexbase.new_attribute('luatestattr')}
137 \luatexbase@directlua{assert(luatexbase.attributes.luatestattr)}
138 \begingroup
139 \escapechar64
```

```
140 \newluatexattribute\anotherattr
141 \endgroup
142 \setluatexattribute\anotherattr{1}
143 \luatexbase@directlua{assert(luatexbase.attributes.anotherattr)}
144 </testplain, testlatex>
145 <testplain>\bye
146 <testlatex>\stop
```