The luatexbase-attr package

Manuel Pégourié-Gonnard
mpg@elzevir.fr

Élie Roux
elie.roux@telecom-bretagne.eu

v0.2a 2010-05-27

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**
1.1 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 **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: `\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.

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.atrribute.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.foobar` (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_attributes(`⟨*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  % $ $
```

```
  \y  39 12 % '
  \y  40 12 % (
  \y  41 12 % )
  \y  42 12 % *
  \y  43 12 % +
  \y  44 12 % ,
  \y  45 12 % -
  \y  46 12 % .
  \y  47 12 % /
  \y  60 12 % <
  \y  61 12 % =
  \y  64 11 % @ (letter)
  \y  62 12 % >
  \y  95 12 % _ (other)
  \y  96 12 % `
  \edef\y#1{\endgroup\edef#1{\the\toks0\relax}\x}%
\expandafter\y\csname lltxb@attr@AtEnd\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}[2010/05/27 v0.2a 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 PackageWarningNoLine\endcsname\relax
      \def\x#1#2{\begingroup\newlinechar10
        \immediate\write16{Package #1 warning: #2}\endgroup}
    \else
      \let\x\PackageWarningNoLine
    \fi
  \expandafter\endgroup
  \x{luatexbase-attr}{LuaTeX is required for this package. Aborting.}
  \lltxb@attr@AtEnd
  \expandafter\endinput
\fi
```

2.1.2 Primitives needed

Load luatexbase-compat.

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

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

```
\begingroup\expandafter\expandafter\expandafter\endgroup
\expandafter\ifx\csname RequirePackage\endcsname\relax
  \input luatexbase-loader.sty
\else
  \RequirePackage{luatexbase-loader}
\fi
\luatexbase@directlua{require('luatexbase.attr')}
```

2.2 User macros

The allocaton 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: wrapper around the Lua function.

```
\def\unsetluatexattribute#1{%
  \begingroup\escapechar\m@ne
  \luatexbase@directlua{%
    luatexbase.unset_attribute("\luatexluaescapestring{\string#1}")}%
  \endgroup}
```

And now the trivial helper macro.

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

That's all folks!

```
\lltxb@attr@AtEnd
```
⟨/texpackage⟩

2.3 Lua module

⟨*luamodule⟩

```
module('luatexbase', package.seeall)
```

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

```
attributes = {}
```

The allocaton 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.

```
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
    last_alloc = last_alloc + 1
    attributes[name] = last_alloc
    unset_attribute(name)
    if not silent then
        texio.write_nl('log', string.format(
            'luatexbase.attributes[%q] = %d', name, last_alloc))
    end
    return last_alloc
end
```

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

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

⟨/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`.

```
⟨testplain⟩\input luatexbase-attr.sty
⟨testlatex⟩\RequirePackage{luatexbase-attr}
⟨*testplain, testlatex⟩
\newluatexattribute\testattr
\setluatexattribute\testattr{1}
\unsetluatexattribute\testattr
\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
```