luaalignstate — Access the alignment counter in LuaTEX

Ruiyuan Chen

2026/11/15 v1.0

In TEX, the alignment brace counter, also known as "master counter" (according to the TEXbook) or `align_state` (according to the source code), is an internal variable that is used to track the balance between all braces `{` and `}` seen so far, and plays a subtle role in the handling of `\halign`/`\valign` and derived macros such as LATEX's `tabular`. The variable is not exposed to the macro language or even to the Lua libraries in LuaTEX. This makes certain rare bugs difficult to track down,[1] and also prevents some useful alignment tricks that would otherwise be possible.[2]

This package provides a hack in LuaTEX to directly inspect and modify the alignment brace counter. There is an underlying format-independent Lua module, as well as a LATEX3 programming wrapper interface.

1 Lua interface

The Lua module is loaded with

```
local luaalignstate = require"luaalignstate"
```

which returns a table containing the following functions.

`luaalignstate.getset()`

```
new, old = luaalignstate.getset(function (old_alignstate)
    return new_alignstate
end)
```

The main interface. Determines the current alignment counter, and optionally updates it in one go. If the provided function is or returns `nil`, no update is made.

TEXhackers note: This function works by executing an empty `\halign`. As such, it *must not be called* while scanning the preamble of another alignment (via e.g., `\span\directlua`); doing so would lead to memory corruption.

Because of this, the function will throw a fatal error if it detects that it was likely called from an alignment preamble, namely if `align_state` < -500000. (This will also happen outside of alignment preambles, if you for some reason set `align_state` to a value less than -500000.)

`luaalignstate.get()`

```
alignstate = luaalignstate.get()
```

`luaalignstate.set()`

```
new, old = luaalignstate.set(new_alignstate)
```

`luaalignstate.add()`

```
new, old = luaalignstate.add(increment)
```

Simple wrappers around `getset()`.

[1] See e.g., https://github.com/latex3/latex2e/issues/1468.

[2] For example, it is difficult to create a fully general mechanism that allows a colspan to be specified in a table cell other than at the very beginning.

2 LaTeX3 interface

The LaTeX package is loaded with

```
\RequirePackage{luaalignstate}
```

This loads the Lua module globally, and defines the following L3 programming functions.

`\luaalignstate_value:` ⋆ Expands in one step to the current value of the alignment counter (in decimal).

TEXhackers note: As with the Lua functions, this is *not* safe to expand (via `\span`) while scanning the preamble of an alignment.

Note that `\luaalignstate_value:` can be somewhat tricky to expand correctly in the desired ambient brace level: for example,

```
\tl_set:No \l_my_tl {\luaalignstate_value:}
```

would likely set `\l_my_tl` to one or two more than the alignment counter value at the point that it is executed, depending on the implementation of `\tl_set:No`.

For this reason, the following getter functions are provided as an alternative.

`\luaalignstate_get:N` `\luaalignstate_get:N` ⟨*tl var*⟩

Sets the ⟨*tl var*⟩ (locally) to the current value of the alignment counter.

`\luaalignstate_use:n` ⋆ `\luaalignstate_use:n {`⟨*code*⟩`}`

Expands in two steps to ⟨*code*⟩ followed by a brace group containing the current value of the alignment counter.

`\luaalignstate_use_inline:n` ⋆ `\luaalignstate_use_inline:n {`⟨*inline function*⟩`}`

Expands in two steps to ⟨*inline function*⟩ applied to the alignment counter value.

`\luaalignstate_set:n` `\luaalignstate_set:n {`⟨*int expr*⟩`}`

Sets the alignment counter to the given ⟨*int expr*⟩.

TEXhackers note: The ⟨*int expr*⟩ may refer to the previous alignment counter value via `\luaalignstate_value:`, which within the ⟨*int expr*⟩ expands to the value at the point where `\luaalignstate_set:n` is called (i.e., outside any braces surrounding or part of ⟨*int expr*⟩).

On the other hand, the ⟨*int expr*⟩ should not refer directly to `\currentgrouplevel`, `\currentgrouptype`, or `\lastnodetype`, as their values will be wrong.

`\luaalignstate_add:n` `\luaalignstate_add:n {`⟨*int expr*⟩`}`

Adds ⟨*int expr*⟩ to the alignment counter.

TEXhackers note: This may be expressed in terms of `\luaalignstate_set:n`, but is ≈ 2 ops faster. Unlike above, here the ⟨*int expr*⟩ should *not* refer to `\luaalignstate_value:`.

3 Implementation

```
⟨*lua⟩
⟨@@=luaalignstate⟩
local luaalignstate = {
    module_info = {
        name        = "luaalignstate",
        description = "Access the alignment counter in LuaTeX",
        date        = "2026/11/15",
        version     = "1.0",
    },
}
if luatexbase then luatexbase.provides_module(luaalignstate.module_info) end
```

3.1 Global imports

Import needed global functions.

```
local pairs = pairs
local select = select
```

Import LuaTEX modules.

```
local luatex = tex
local luatex_token = token
local luatex_node = node
local luatex_status = status
_ENV = nil
```

Import commonly used functions from modules.

```
local tex_getnest = luatex.getnest
local tex_runtoks = luatex.runtoks
local tex_quittoks = luatex.quittoks
local token_command_id = luatex_token.command_id
local token_new = luatex_token.new
local token_get_next = luatex_token.get_next
local token_scan_code = luatex_token.scan_code
local token_put_next = luatex_token.put_next
local token_unchecked_put_next = luatex_token.unchecked_put_next
local node_direct_new = luatex_node.direct.new
local node_direct_free = luatex_node.direct.free
local node_direct_getnext = luatex_node.direct.getnext
local node_direct_getdata = luatex_node.direct.getdata
local node_direct_setnext = luatex_node.direct.setnext
local node_direct_setdata = luatex_node.direct.setdata
local node_direct_setfield = luatex_node.direct.setfield
```

3.2 Constants

`EQ_...` Some hardcoded eqtb locations (from `equivalents.h`).

```
local EQ_frozen_control_sequence = ((1 +1)+luatex_status.hash_size)
local EQ_local_base = ((((((EQ_frozen_control_sequence+13 )+(5535-0 +1))+1)+19)+65536)+65536)
```

(*End of definition for* `EQ_...`)

TOKEN_... Commonly used tokens.

```
local CS_TOKEN_FLAG = 0x1FFFFFFF
local COMMAND_lua_local_call = token_command_id"lua_local_call"
local TOKEN_bgroup = token_new(("{"):byte(), 1)
local TOKEN_egroup = token_new(("}"):byte(), 2)
local TOKEN_param = token_new(("#"):byte(), 6)
local TOKEN_immediateassignment = token_new(5, token_command_id"convert")
local TOKEN_nonstopmode = token_new(1, token_command_id"set_interaction")
local TOKEN_halign = token_new(0, token_command_id"halign")
local TOKEN_span = token_new(0x110002, 4)
```

From `equivalents.h`. (NOTE: `\cr` must be a named control sequence, not a nameless internal token, since the internal token is treated as `out_param` when read.)

```
local TOKEN_cr = token_new(CS_TOKEN_FLAG + EQ_frozen_control_sequence + 1, 0)
local TOKEN_endv = token_new(CS_TOKEN_FLAG + EQ_frozen_control_sequence + 6, 0)
local TOKEN_everycr = token_new(EQ_local_base + 8, token_command_id"assign_toks")
```

(*End of definition for* `TOKEN_...`.)

MODE_hmode Horizontal and vertical mode constants (in `tex.nest.mode`).
MODE_vmode

```
local MODE_vmode
for k, v in pairs(luatex.getmodevalues()) do
    if v == "vertical" then MODE_vmode = k end
end
```

(*End of definition for* `MODE_hmode` *and* `MODE_vmode`.)

NODEID_... Node types.

```
local NODEID_align_stack = luatex_node.id"align_stack"
```

(*End of definition for* `NODEID_...`.)

3.3 Helpers

tmp_whatsit_ptr A whatsit to help with allocations.

```
local tmp_whatsit_ptr = node_direct_new("whatsit", "user_defined")
```

(*End of definition for* `tmp_whatsit_ptr`.)

new_lualocal Creates a new (expandable) luacall that's stored in the Lua C registry.

```
local function new_lualocal(f)
```

It is necessary to set the whatsit integer value to 0 first, or else setting the lua value will kill any existing registry entry.

```
    node_direct_setfield(tmp_whatsit_ptr, "type", 100)  -- ("d"):byte()
    node_direct_setdata(tmp_whatsit_ptr, 0)
    node_direct_setfield(tmp_whatsit_ptr, "type", 108)  -- ("l"):byte()
    node_direct_setdata(tmp_whatsit_ptr, f)
    node_direct_setfield(tmp_whatsit_ptr, "type", 100)  -- ("d"):byte()
    return token_new(node_direct_getdata(tmp_whatsit_ptr), COMMAND_lua_local_call)
end
```

(*End of definition for* `new_lualocal`.)

fatal_error() Raises a fatal error. We could use any of the several tokens that triggers a `normal_error()`, `fatal_error()`, or `confusion()`. Since this is called when the `align_state` is manipulated from alignment preambles, it seems that the "interwoven alignment preamble" error message is the most appropriate. The `scan_code()` is used to flush the two token lists inserted by `runtoks()` from the error context. (Unfortunately, there seems to be no way to also flush the pushed `\endtemplate`.)

```
local function fatal_error(msg, help)
    return tex_runtoks(function ()
        token_get_next()
        token_put_next(TOKEN_endv)
        if luatex.interactionmode > 1 then
            token_put_next(TOKEN_immediateassignment, TOKEN_nonstopmode)
        end
        token_scan_code(0)
        return luatex.error(msg, help or {})
    end)
end
```

(*End of definition for* `fatal_error()`.)

3.4 Getting and setting the alignment counter

Now for the main hack. The strategy is to execute (the equivalent of)

```
⟨determine align_stack location⟩
\halign{%
    \span\immediateassignment\everycr{}%
    #\cr
    ⟨read/write align_state from align_stack⟩
}
```

When LuaTEX executes `\halign`, it creates a node of type `align_stack` (see `align.c:push_alignment()`), which contains a field that records the `align_state` before the alignment began, which is restored after the alignment ends. Thus, the goal of the expandable luacall ⟨*read/write* `align_state` *from* `align_stack`⟩ is to access this field. Note that because it is expandable, and we nullified the `\everycr` (*inside* the outer `\halign` group, but outside of any cell), the alignment won't have any rows: everything takes place while `align_peek()` is looking for the first row, only to find the end-brace `}`.

Unfortunately, the `align_stack` does not seem to be referenced from any nodes in the current list, so we cannot directly access it. Instead we take advantage of the way that node allocation works in LuaTEX: right before the `\halign`, we allocate a `align_stack` ourselves, record its address, and then free it, putting it onto the free list and ensuring that `\halign` will allocate in the same location. (So this is an intentional use-after-free!)

We export a main public function, which needs to pass arguments to several auxiliaries via upvariables; we isolate these inside a `do` block for clarity.

```
local getset do
```

getset_align_stack_ptr The location of the `align_stack` node (an integer/`node.direct` pointer), determined right before the `\halign` as described above.

```
local getset_align_stack_ptr
```

(*End of definition for* `getset_align_stack_ptr`.)

getset_func
getset_oldval
getset_newval

Used to pass the user-provided getset function and the old and new `align_state` values.

```
local getset_func, getset_oldval, getset_newval
```

(*End of definition for* `getset_func`, `getset_oldval`, *and* `getset_newval`.)

getset_do

Executed expandably inside of the `\halign`, right before it ends (without any rows). Fetch the enclosing `align_state` from the `align_stack` node, run the caller-provided update function, and set the `align_state` to the new value if different.

```
local function getset_do()
```

The following calls `vlink(getset_align_stack_ptr + 3)`, which stores the enclosing `align_state`; see `lnodelib.c`:`lua_nodelib_direct_getnext()` and `align.c`:`push_alignment()`.

```
    local oldval = node_direct_getnext(getset_align_stack_ptr + 3) or 0
    if oldval < -500000 then
        return fatal_error(
            "luaalignstate Fatal Error: Cannot access from alignment preamble",
            {
                "Accessing the alignment counter from a preamble causes memory corruption.",
                "I'm going to quit now.",
            }
        )
    end
    getset_oldval = oldval
    local newval = getset_func and getset_func(oldval)
    if newval == nil then
        getset_newval = oldval
    else
        if newval ~= oldval then
            node_direct_setnext(getset_align_stack_ptr + 3, newval)
        end
        getset_newval = newval
    end
    token_put_next(TOKEN_egroup)
    token_scan_code(0)
    return tex_quittoks()
end
local TOKEN_getset_do = new_lualocal(getset_do)
```

(*End of definition for* `getset_do`.)

TOKENS_getset

The token list containing `\halign`, sans the close brace which is inserted by `getset_do()`.

```
local TOKENS_getset = {
    TOKEN_halign, TOKEN_bgroup,
        TOKEN_span, TOKEN_immediateassignment, TOKEN_everycr, TOKEN_bgroup, TOKEN_egroup,
        TOKEN_param, TOKEN_cr,
        TOKEN_getset_do,
}
```

(*End of definition for* `TOKENS_getset`.)

getset_run

The function that's executed by `tex.runtoks()`.

```
local function getset_run()
```

Remove the `\endlocalcontrol`. This allows the user function to scan the token stream (see e.g., `\__luaalignstate_set:w`), which is also why we left off the close brace above.

```
    token_get_next()
```

Set the mode to internal vertical, to prevent `\halign` from complaining. The enclosing mode is already remembered and restored by `runtoks()`.

```
    tex_getnest().mode = -MODE_vmode
```

Now put an `align_stack` on the free list, record its address, and insert the `halign` tokens.

```
    getset_align_stack_ptr = node_direct_new(NODEID_align_stack)
    node_direct_free(getset_align_stack_ptr)
    token_unchecked_put_next(TOKENS_getset)
end
```

(*End of definition for* `getset_run`.)

`luaalignstate.getset()` The actual main function. The argument should take the current `align_state` as input, and return the modified value (if any).

```
function getset(func)
    getset_func = func
    tex_runtoks(getset_run)
    getset_func = nil
    return getset_newval, getset_oldval
end
luaalignstate.getset = getset
end
```

(*End of definition for* `luaalignstate.getset()`. *This function is documented on page* 1.)

`luaalignstate.get()` `luaalignstate.set()` `luaalignstate.add()` Simple wrappers.

```
function luaalignstate.get() return select(1, getset()) end
do
local newval
local function retnewval()
    return newval
end
local function addvals(oldval)
    return oldval + newval
end
function luaalignstate.set(s)
    newval = s
    return getset(retnewval)
end
function luaalignstate.add(s)
    newval = s
    return getset(addvals)
end
end
```

(*End of definition for* `luaalignstate.get()`, `luaalignstate.set()`, *and* `luaalignstate.add()`. *These functions are documented on page* 1.)

Clean up and return.

```
node_direct_setfield(tmp_whatsit_ptr, "type", 100)  -- ("d"):byte()
node_direct_free(tmp_whatsit_ptr)
return luaalignstate
```

⟨/lua⟩

3.5 LaTeX package

```
⟨*package⟩
\ProvidesExplPackage
    {luaalignstate}
    {2026/11/15}
    {1.0}
    {Access the alignment counter in LuaTeX}
\sys_if_engine_luatex:F {
    \PackageError {luaalignstate} {LuaTeX~required} {}
    \file_input_stop:
}
```

In LaTeX, we create the Lua functions the standard way, using `luatexbase`. To keep the Lua module LaTeX-independent, we settle for a short block of inline Lua code.

```
\group_begin: \use:n {
    \char_set_catcode_space:n {32}
    \char_set_catcode_other:N \\
    \use:nnn
\group_end:
\lua_now:n}{{
    local tostring = tostring
    local tex_write = tex.write
    local token_scan_int = token.scan_int
    local token_set_lua = token.set_lua
    local token_set_macro = token.set_macro
    local lua_functions_table = lua.get_functions_table()
    local luatexbase_new_luafunction = luatexbase.new_luafunction
    luaalignstate = require"luaalignstate"
    local luaalignstate_getset = luaalignstate.getset
    local luaalignstate_add = luaalignstate.add
    _ENV = nil
    local function luacmd(name, f, ...)
        local id = luatexbase_new_luafunction(name)
        lua_functions_table[id] = f
        token_set_lua(name, id, "global", ...)
    end
```

`\luaalignstate_value:` The getter is straightforward.

```
    luacmd("luaalignstate_value:", function ()
        return tex_write(tostring(luaalignstate_getset()))
    end)
```

(*End of definition for* `\luaalignstate_value:`*. This function is documented on page* 2.)

`\__luaalignstate_set:w` `\__luaalignstate_set:w` ⟨*integer*⟩
`\__luaalignstate_add:w` `\__luaalignstate_add:w` ⟨*integer*⟩
`\luaalignstate_set:n`
`\luaalignstate_add:n`

For the setters, we make a luacall that uses primitive-like syntax, and a macro wrapper. The setter uses `getset()` rather than `set()`, to support accessing the old value during evaluation of the new value, by locally redefining `\luaalignstate_value:` (within the `\halign` group) to a token list containing the old value.

```
    local function set_scan(old_alignstate)
        token_set_macro("luaalignstate_value:", tostring(old_alignstate))
```

```
        return token_scan_int()
    end
    luacmd("__luaalignstate_set:w", function ()
        return luaalignstate_getset(set_scan)
    end, "protected")
```

Note that `\__luaalignstate_set:w` doesn't scan its argument until *inside* the `\halign`, whereas `\__luaalignstate_add:w` scans its argument immediately, before initiating the `\halign`.

```
    luacmd("__luaalignstate_add:w", function ()
        return luaalignstate_add(token_scan_int())
    end, "protected")
}}%
\cs_new_protected:Npn \luaalignstate_set:n #1 {
    \__luaalignstate_set:w \tex_numexpr:D #1 \scan_stop:
}
\cs_new_protected:Npn \luaalignstate_add:n #1 {
    \__luaalignstate_add:w \tex_numexpr:D #1 \scan_stop:
}
```

(*End of definition for* `\__luaalignstate_set:w` *and others. These functions are documented on page* 2.)

`\luaalignstate_get:N`
`\luaalignstate_use:n`
`\luaalignstate_use_inline:n`

For the in-context getters, we expand `\luaalignstate_value:` using brace tricks to balance the alignment counter.

```
\cs_new_protected:Npn \luaalignstate_get:N #1 {
    \tex_edef:D #1 { \if_false: } \fi: \luaalignstate_value: \if_false: { \fi: }
}
\cs_new:Npn \luaalignstate_use:n #1 {
    \tex_expanded:D {
        \tex_unexpanded:D {#1}
        { \if_false: }} \fi: \luaalignstate_value: \if_false: {{ \fi: }
    }
}
\cs_new:Npn \luaalignstate_use_inline:n #1 {
    \exp:w
    \tex_immediateassignment:D \cs_gset_nopar:Npn \__luaalignstate_tmp:w ##1 {\exp_end: #1}
    \exp_after:wN \__luaalignstate_tmp:w \tex_expanded:D {
        { \if_false: }} \fi: \luaalignstate_value: \if_false: {{ \fi: }
    }
}
```

(*End of definition for* `\luaalignstate_get:N`, `\luaalignstate_use:n`, *and* `\luaalignstate_use_inline:n`. *These functions are documented on page* 2.)

```
⟨/package⟩
```