The `luakeyval` Module

Version: 0.1, 2025-12-01

Udi Fogiel, 2025

The luakeyval LuTeX module is a minimal key/value parser for LuaTeX formats based on `token.scan_key_cs`. As such, the keyword parsing is similar to how TEX parses keywords in special primitives like `\hrule`.

Unlike TEX's scanner, `luakeyval` scans key/val pairs inside braces. This avoids the need to append a `\relax` to the key/val list, which would otherwise be required to stop TEX from scanning too far and potentially creating expansion problems.

1 Usage

The module provides the `process` function, which accepts a table of keys, and a table of error messages.

Each key in the table should have a table of parameters as a value. The possible parameters are

- **scanner:** a function that will scan the value of the key from TEX to Lua. The value will usually be a function from LuaTEX's token library, but you can use your own.
- **args:** a table of arguments that will be passed to the scanner function.
- **default:** default: a value returned if the key is not followed by a `=`.
- **func:** a function that will be executed each time the key appears.

None of the parameters is mandatory, but a key must have at least one of `default` or `scanner`.

The error messages table can have the following entries

- **error1:** a message that will be displayed if something went wrong while processing a key/val list.
- **error2:** a message that will be displayed after `error1` in case a user press the `H` key for more information.
- **value_required:** a message that will be displayed if no value given to a key (when there is no `=` after the key) and the key does not have a default value.
- **value_forbidden:** a message that will be displayed if a key was given a value and the key does not have a `scanner` function.

2 Example

The following code defines the `\coloredrule` macro, which accepts the keys `width`, `height`, `depth` and `color`, and prints a colored rule according to the values given.

```
local keyval = require('luakeyval')
local rule_keys = {
    width = {scanner = token.scan_dimen, default = tex.sp('1cm')},
    height = {scanner = token.scan_dimen, default = tex.sp('1ex')},
    depth = {scanner = token.scan_dimen, default = 0},
    color = {scanner = token.scan_string},
}

local messages = {
```

```
    error1 = "colored rule: Wrong syntax in \\coloredrule",
    value_forbidden = 'colored rule: The key "%s" does not accept a value',
    value_required = 'colored rule: The key "%s" requires a value',
}

local function make_rule()
    local opts = keyval.process(rule_keys, messages)
    local rule = node.new('rule')
    rule.width = opts.width or tex.sp('1cm')
    rule.height = opts.height or tex.sp('1ex')
    rule.depth = opts.depth or 0
    local color_start = node.new('whatsit', 'pdf_literal')
    color_start.mode = 0
    color_start.data = opts.color .. " rg"
    local color_end = node.new('whatsit', 'pdf_literal')
    color_end.mode = 0
    color_end.data = '0 g'
    rule.next = color_end
    color_start.next = rule
    rule = color_start
    node.write(rule)
end

token.set_lua('coloredrule', #lua.get_functions_table() +1, 'protected')
lua.get_functions_table()[#lua.get_functions_table()+1] = make_rule
```

Now `\coloredrule{width = 10pt height = 5pt color={1 0 0}}` prints

3 Important Limitations

Since the key/val parser is only a minimal layer on top of `token.scan_keyword`, key/val pairs should be separated with spaces, not commas, and avoid defining a key that is a prefix of another key (unless you control the scanning order, see Section 4).

4 Scanning Order

The `process` function loops over the keys table using LuaTeX's `token.scan_key_cs`. Since Lua tables do not guarantee key order when iterating with `pairs()`, the scanning order is not deterministic if you just provide a plain table.

This is important when one key is a prefix of another (for example, `long` and `longer`). If the shorter key is checked first, it may match part of the input and leave extra characters unprocessed, leading to errors.

To control the scanning order explicitly, pass a third argument to `process`: a list of keys in the exact order they should be scanned. This also allows scanning only a subset of keys if desired.

```
local keys = {
  long   = {scanner = token.scan_string},
  longer = {scanner = token.scan_string},
}

-- Default scan (order not guaranteed)
local opts = keyval.process(keys, messages)
-- input: {longer="test"} may incorrectly match 'long' first

-- Controlled scan with explicit order
local order = {"longer", "long"}
local opts = keyval.process(keys, messages, order)
```

5 Implementation

The full Lua implementation is shown below for reference.

luakeyval.lua

```lua
-- luakeyval Version: 0.1, 2025-12-01

local put_next = token.unchecked_put_next
local get_next = token.get_next
local scan_toks = token.scan_toks
local scan_keyword = token.scan_keyword_cs

local texerror, utfchar = tex.error, utf8.char
local format = string.format

-- local relax = token.new(token.biggest_char() + 1)
local relax
do
-- initialization of the new primitives.
  local prefix = '@lua^key&val_' -- unlikely prefix...
  while token.is_defined(prefix .. 'relax') do
    prefix = prefix .. '@lua^key&val_'
  end
  tex.enableprimitives(prefix,{'relax'})
-- Now we create new tokens with the meaning of
-- the primitives.
  local tok = token.create(prefix .. 'relax')
  relax = token.new(tok.mode, tok.command)
end

local function check_delimiter(error1, error2, key)
    local tok = get_next()
    if tok.tok ~= relax.tok then
        local tok_name = tok.csname or utfchar(tok.mode)
        texerror(format(error1, key, tok_name),{format(error2, key, tok_name)})
        put_next({tok})
    end
end

local unpack = table.unpack
local function process_keys(keys, messages, order)
    assert(type(keys) == 'table')
    local matched, vals, curr_key = true, { }
    messages = messages or { }
    local value_forbidden = messages.value_forbidden
        or 'luakeyval: The key "%s" does not accept a value'
    local value_required = messages.value_required
        or 'luakeyval: The key "%s" requires a value'
    local error1 = messages.error1
        or 'luakeyval: Wrong syntax when processing keys'
    local error2 = messages.error2
        or 'luakeyval: The last scanned key was "%s".\nUnexpected token "%s" encountered.'
    local key_list = { }
    if order then
        for _, k in ipairs(order) do
            key_list[#key_list+1] = k
        end
    else
        for k in pairs(keys) do
            key_list[#key_list+1] = k
        end
    end
    local toks = scan_toks()
    toks[#toks+1] = relax
    put_next(toks)
    while matched do
        matched = false
        for _, key in ipairs(key_list) do
            local param = keys[key]
            if scan_keyword(key) then
```

```
                matched = true
                curr_key = key
                local args = param.args or { }
                local scanner = param.scanner
                local val
                if scan_keyword('=') then
                    if scanner then
                        val = scanner(unpack(args))
                    else
                        texerror(format(value_forbidden, key))
                    end
                else
                    val = param.default
                    if val == nil then
                        texerror(format(value_required, key))
                    end
                end
                local func = param.func
                if func then func(key,val) end
                vals[key] = val
                break
            end
        end
    end
    check_delimiter(error1, error2, curr_key or '<none>')
    return vals
end

local function scan_choice(...)
    local choices = {...}
    for _, choice in ipairs(choices) do
        if scan_keyword(choice) then
            return choice
        end
    end
    return nil
end

local function scan_bool()
    if scan_keyword('true') then
        return true
    elseif scan_keyword('false') then
        return false
    end
    return nil
end

return {
    process = process_keys,
    choices = scan_choice,
    bool = scan_bool,
}
```