The luakeys package

Josef Friedrich
josef@friedrich.rocks
github.com/Josef-Friedrich/luakeys

v0.5 from 2022/04/04

```
local luakeys = require('luakeys')
local kv = luakeys.parse('level1={level2={level3={dim=1cm,bool=true,num=-
↪  0.001,str=lua}}}')
luakeys.print(kv)
```

Result:

```
{
  ['level1'] = {
    ['level2'] = {
      ['level3'] = {
        ['dim'] = 1864679,
        ['bool'] = true,
        ['num'] = -0.001
        ['str'] = 'lua',
      }
    }
  }
}
```

Contents

1 Introduction

`luakeys` is a Lua module that can parse key-value options like the TEX packages keyval, kvsetkeys, kvoptions, xkeyval, pgfkeys etc. do. `luakeys`, however, accompilshes this task entirely, by using the Lua language and doesn't rely on TEX. Therefore this package can only be used with the TEX engine LuaTEX. Since `luakeys` uses LPeg, the parsing mechanism should be pretty robust.

The TUGboat article "Implementing key–value input: An introduction" (Volume 30 (2009), No. 1) by Joseph Wright and Christian Feuersänger gives a good overview of the available key-value packages.

This package would not be possible without the article Parsing complex data formats in LuaTEX with LPEG (Volume 40 (2019), No. 2).

2 Usage

2.1 Using the Lua module `luakeys.lua`

The core functionality of this package is realized in Lua. So you can use `luakeys` without using the wrapper TEX files `luakeys.sty` and `luakeys.tex`.

```
\documentclass{article}
\directlua{
  luakeys = require('luakeys')
}

\newcommand{\helloworld}[2][]{
  \directlua{
    local keys = luakeys.parse('\luaescapestring{\unexpanded{#1}}')
    luakeys.print(keys)
    local marg = '#2'
    tex.print(keys.greeting .. ', ' .. marg .. keys.punctuation)
  }
}
\begin{document}
\helloworld[greeting=hello,punctuation=!]{world}
\end{document}
```

2.2 Using the LuaLATEX wrapper `luakeys.sty`

The supplied LuaLATEX file is quite small:

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{luakeys}
\directlua{luakeys = require('luakeys')}
```

It loads the Lua module into the global variable `luakeys`.

```
\documentclass{article}
\usepackage{luakeys}

\begin{document}
  \directlua{
    local keys = luakeys.parse('one,two,three')
    tex.print(keys[1])
    tex.print(keys[2])
    tex.print(keys[3])
  }
\end{document}
```

2.3 Using the plain LuaTEX wrapper `luakeys.tex`

Even smaller is the file `luakeys.tex`. It consists of only one line:

```
\directlua{luakeys = require('luakeys')}
```

It does the same as the LuaLATEX wrapper and loads the Lua module `luakeys.lua` into the global variable `luakeys`.

```
\input luakeys.tex

\directlua{
  local keys = luakeys.parse('one,two,three')
  tex.print(keys[1])
  tex.print(keys[2])
  tex.print(keys[3])
}
\bye
```

3 Syntax of the recognized key-value format

3.1 A attempt to put the syntax into words

A key-value pair is definied by an equal sign (`key=value`). Several key-value pairs or values without keys are lined up with commas (`key=value,value`) and build a key-value list. Curly brackets can be used to create a recursive data structure of nested key-value lists (`level1={level2={key=value,value}}`).

3.2 An (incomplete) attempt to put the syntax into the Extended Backus-Naur Form

⟨*list*⟩ ::= { ⟨*list-item*⟩ }

⟨*list-container*⟩ ::= '`{`' ⟨*list*⟩ '`}`'

⟨*list-item*⟩ ::= (⟨*list-container*⟩ | ⟨*key-value-pair*⟩ | ⟨*value*⟩) ['`,`']

⟨*key-value-pair*⟩ ::= ⟨*value*⟩ '`=`' (⟨*list-container*⟩ | ⟨*value*⟩)

⟨*value*⟩ ::= ⟨*boolean*⟩
| ⟨*dimension*⟩
| ⟨*number*⟩
| ⟨*string-quoted*⟩
| ⟨*string-unquoted*⟩

⟨*sign*⟩ ::= '`-`' | '`+`'

⟨*integer*⟩ ::= '`0`' | '`1`' | '`2`' | '`3`' | '`4`' | '`5`' | '`6`' | '`7`' | '`8`' | '`9`'

⟨*unit*⟩ ::= '`bp`' | '`BP`'
| '`cc`' | '`CC`'
| '`cm`' | '`CM`'
| '`dd`' | '`DD`'
| '`em`' | '`EM`'
| '`ex`' | '`EX`'
| '`in`' | '`IN`'
| '`mm`' | '`MM`'
| '`nc`' | '`NC`'
| '`nd`' | '`ND`'
| '`pc`' | '`PC`'

| ‘pt’ | ‘PT’
| ‘sp’ | ‘SP’

⟨*boolean*⟩ ::= ⟨*boolean-true*⟩ | ⟨*boolean-false*⟩

⟨*boolean-true*⟩ ::= ‘`true`’ | ‘`TRUE`’ | ‘`True`’

⟨*boolean-false*⟩ ::= ‘`false`’ | ‘`FALSE`’ | ‘`False`’

... to be continued

3.3 Recognized data types

3.3.1 boolean

The strings `true`, `TRUE` and `True` are converted into Lua's boolean type `true`, the strings `false`, `FALSE` and `False` into `false`.

```
\luakeysdebug{
  lower case true = true,
  upper case true = TRUE,
  title case true = True
  lower case false = false,
  upper case false = FALSE,
  title case false = False,
}
```

```
{
  ['lower case true'] = true,
  ['upper case true'] = true,
  ['title case true'] = true,
  ['lower case false'] = false,
  ['upper case false'] = false
  ['title case false'] = false,
}
```

3.3.2 number

```
\luakeysdebug{
  num1 = 4,
  num2 = -4,
  num3 = 0.4
}
```

```
{
  ['num1'] = 4,
  ['num2'] = -4,
  ['num3'] = 0.4
}
```

3.3.3 dimension

`luakeys` detects TEX dimensions and automatically converts the dimensions into scaled points using the function `tex.sp(dim)`. Use the option `convert_dimensions` of the function `parse(kv_string, options)` to disalbe the automatic conversion.

```
local result = parse('dim=1cm', {
  convert_dimensions = false,
})
```

If you want to convert a scale point into a unit string you can use the module lualibs-util-dim.lua.

```
\begin{luacode}
require('lualibs')
tex.print(number.todimen(tex.sp('1cm'), 'cm', '%0.0F%s'))
\end{luacode}
```

Unit name	Description
bp	big point
cc	cicero
cm	centimeter
dd	didot
em	horizontal measure of M
ex	vertical measure of x
in	inch
mm	milimeter
nc	new cicero
nd	new didot
pc	pica
pt	point
sp	scaledpoint

```
\luakeysdebug{
  bp = 1bp,
  cc = 1cc,
  cm = 1cm,
  dd = 1dd,
  em = 1em,
  ex = 1ex,
  in = 1in,
  mm = 1mm,
  nc = 1nc,
  nd = 1nd,
  pc = 1pc,
  pt = 1pt,
  sp = 1sp,
}
```

```
{
  ['bp'] = 65781,
  ['cc'] = 841489,
  ['cm'] = 1864679,
  ['dd'] = 70124,
  ['em'] = 655360,
  ['ex'] = 282460,
  ['in'] = 4736286,
  ['mm'] = 186467,
  ['nc'] = 839105,
  ['nd'] = 69925,
  ['pc'] = 786432,
  ['pt'] = 65536,
  ['sp'] = 1,
}
```

3.3.4 string

There are two ways to specify strings: With or without quotes. If the text have to contain commas or equal signs, then double quotation marks must be used.

```
\luakeysdebug{
  without quotes = no commas and
  ↪  equal signs are allowed,
  with double quotes = ", and = are
  ↪  allowed",
}
```

```
{
  ['without quotes'] = 'no commas
  ↪  and equal signs are allowed',
  ['with double quotes'] = ', and =
  ↪  are allowed',
}
```

3.3.5 Standalone values

Standalone values are values without a key. They are converted into an array. In Lua an array is a table with numeric indexes (The first index is 1).

```
\luakeysdebug{one,two,three}
```

```
{ 'one', 'two', 'three' }
```

is equivalent to

```
{
  [1] = 'one',
  [2] = 'two',
  [3] = 'three',
}
```

All recognized data types can be used as standalone values.

```
\luakeysdebug{one,2,3cm}
```

```
{ 'one', 2, 5594039 }
```

4 Exported functions of the Lua module `luakeys.lua`

To learn more about the individual functions (local functions), please read the source code documentation, which was created with LDoc. The Lua module exports this functions:

```
local luakeys = require('luakeys')
local parse = luakeys.parse
local render = luakeys.render
--local print = luakeys.print -- That would overwrite the built-in Lua function
local save = luakeys.save
local get = luakeys.get
```

4.1 `parse(kv_string, options)`: table

The function `parse(input_string, options)` is the main method of this module. It parses a key-value string into a Lua table.

```
\newcommand{\mykeyvalcmd}[1][]{
  \directlua{
    result = luakeys.parse('#1')
    luakeys.print(result)
  }
  #2
}
\mykeyvalcmd[one=1]{test}
```

In plain TEX:

```
\def\mykeyvalcommand#1{
  \directlua{
    result = luakeys.parse('#1')
    luakeys.print(result)
  }
}
\mykeyvalcmd{one=1}
```

The function can be called with an options table. This options are supported:

```
local result = parse('one,two,three', {
  convert_dimensions = false,
  unpack_single_array_value = false,
  standalone_as_true = false,
  converter = function(key, value, depth, current_table, root_table)
    return key, value
  end,
  case_insensitive_keys = false,
})
```

The options can also be set globally using the exported table `default_options`:

```
luakeys.parse('dim=1cm') -- {dim = 1864679}
luakeys.default_options.convert_dimensions = false
-- or:
-- local defaults = luakeys.default_options
-- defaults.convert_dimensions = false
```

```
luakeys.parse('dim=1cm') -- {dim = '1cm'}
```

4.2 render(tbl): string

The function `render(tbl)` reverses the function `parse(kv_string)`. It takes a Lua table and converts this table into a key-value string. The resulting string usually has a different order as the input table.

```
result = luakeys.parse('one=1,two=2,tree=3,')
print(luakeys.render(result))
--- one=1,two=2,tree=3,
--- or:
--- two=2,one=1,tree=3,
--- or:
--- ...
```

In Lua only tables with 1-based consecutive integer keys (a.k.a. array tables) can be parsed in order.

```
result = luakeys.parse('one,two,three')
print(luakeys.render(result))
--- one,two,three, (always)
```

4.3 print(tbl): void

The function `print(tbl)` pretty prints a Lua table to standard output (stdout). It is a utility function that can be used to debug and inspect the resulting Lua table of the function `parse`. You have to compile your TEX document in a console to see the terminal output.

```
result = luakeys.parse('level1={level2={key=value}}')
luakeys.print(result)
```

The output should look like this:

```
{
  ['level1'] = {
    ['level2'] = {
      ['key'] = 'value',
  },
}
```

4.4 save(identifier, result): void

The function `save(identifier, result)` saves a result (a table from a previous run of `parse`) under an identifier. Therefore, it is not necessary to pollute the global namespace to store results for the later usage.

4.5 get(identifier): table

The function `get(identifier)` retrieves a saved result from the result store.

5 Debug packages

Two small debug packages are included in `luakeys`. One debug package can be used in LaTeX (luakeys-debug.sty) and one can be used in plain TeX (luakeys-debug.tex). Both packages provide only one command: `\luakeysdebug{kv-string}`

```
\luakeysdebug{one,two,three}
```

Then the following output should appear in the document:

```
{
  [1] = 'one',
  [2] = 'two',
  [3] = 'three',
}
```

5.1 For plain TeX: luakeys-debug.tex

An example of how to use the command in plain TeX:

```
\input luakeys-debug.tex
\luakeysdebug{one,two,three}
\bye
```

5.2 For LaTeX: luakeys-debug.sty

An example of how to use the command in LaTeX:

```
\documentclass{article}
\usepackage{luakeys-debug}
\begin{document}
\luakeysdebug[
  unpack single array values=false,
  convert dimensions=false
]{one,two,three}
\end{document}
```

6 Implementation

6.1 luakeys.lua

```
1  -- luakeys.lua
2  -- Copyright 2021-2022 Josef Friedrich
3  --
4  -- This work may be distributed and/or modified under the
5  -- conditions of the LaTeX Project Public License, either version 1.3c
6  -- of this license or (at your option) any later version.
7  -- The latest version of this license is in
8  --   http://www.latex-project.org/lppl.txt
9  -- and version 1.3c or later is part of all distributions of LaTeX
10 -- version 2008/05/04 or later.
11 --
12 -- This work has the LPPL maintenance status `maintained'.
13 --
14 -- The Current Maintainer of this work is Josef Friedrich.
15 --
16 -- This work consists of the files luakeys.lua, luakeys.sty, luakeys.tex
17 -- luakeys-debug.sty and luakeys-debug.tex.
18
19 --- A key-value parser written with Lpeg.
20 --
21 -- Explanations of some LPeg notation forms:
22 --
23 -- * `patt ^ 0` = `expression *`
24 -- * `patt ^ 1` = `expression +`
25 -- * `patt ^ -1` = `expression ?`
26 -- * `patt1 * patt2` = `expression1 expression2`: Sequence
27 -- * `patt1 + patt2` = `expression1 / expression2`: Ordered choice
28 --
29 -- * [TUGboat article: Parsing complex data formats in LuaTEX with
   ↪  LPEG](https://tug.org/TUGboat/tb40-2/tb125menke-Patterndf)
30 --
31 -- @module luakeys
32
33 local lpeg = require('lpeg')
34 local Variable = lpeg.V
35 local Pattern = lpeg.P
36 local Set = lpeg.S
37 local Range = lpeg.R
38 local CaptureGroup = lpeg.Cg
39 local CaptureFolding = lpeg.Cf
40 local CaptureTable = lpeg.Ct
41 local CaptureConstant = lpeg.Cc
42 local CaptureSimple = lpeg.C
43
44 if not tex then
45   tex = {}
46
47   -- Dummy function for the tests.
48   tex['sp'] = function (input)
49     return 1234567
50   end
51 end
52
53 --- Option handling
54 -- @section
55
56 --- This table stores all allowed option keys.
57 local option_keys = {
```

```
    'convert_dimensions',
    'unpack_single_array_values',
    'standalone_as_true',
    'converter',
    'case_insensitive_keys'
  }

  --- The default options.
  local default_options = {
    convert_dimensions = true,
    unpack_single_array_values = true,
    standalone_as_true = false,
  }

  local function throw_error(message)
    if type(tex.error) == 'function' then
      tex.error(message)
    else
      error(message)
    end
  end

  --- Convert a key so that it can be written as a table field without
  --  quotes and square brackets (for example `one 2` becomes `one_2`).
  --  The key can then reference values from a table using dot notation.
  --  (`table["one 2"]` becomes `table.one_2`).
  --
  -- @tparam string key The key to be converted.
  --
  -- @treturn string The converted key.
  local function luafy_key(key)
    return key:gsub('[^%w]+', '_')
  end

  --- Convert all keys in a table to strings containig only alphanumeric
  -- characters and underscores.
  --
  -- @param raw_options Some raw options.
  --
  -- @treturn table Returns always a table. If the input value is not a
  -- an empty table is returned.
  local function luafy_options(raw_options)
    if type(raw_options) ~= 'table' then
      raw_options = {}
    end
    local options = {}
    for key, value in pairs(raw_options) do
      options[luafy_key(key)] = value
    end
    return options
  end

  --- All option keys can be written with underscores or with spaces as
  -- separators.
  -- For the LaTeX version of the macro
  --  `\luakeysdebug[options]{kv-string}`.
  --
  -- @tparam table options_raw Options in a raw format. The table may be
  -- empty or some keys are not set.
  --
  -- @treturn table
  local function normalize_parse_options (options_raw)
```

```lua
  options_raw = luafy_options(options_raw)
  local options = {}

  for _, option_name in ipairs(option_keys) do
    if options_raw[option_name] ~= nil then
      options[option_name] = options_raw[option_name]
    else
      options[option_name] = default_options[option_name]
    end
  end

  return options
end

--- Parser / Lpeg related
-- @section

--- Generate the PEG parser using Lpeg.
--
-- @treturn userdata The parser.
local function generate_parser(options)
  -- Optional whitespace
  local white_space = Set(' \t\n\r')

  --- Match literal string surrounded by whitespace
  local ws = function(match)
    return white_space^0 * Pattern(match) * white_space^0
  end

  local capture_dimension = function (input)
    if options.convert_dimensions then
      return tex.sp(input)
    else
      return input
    end
  end

  --- Add values to a table in two modes:
  --
  -- # Key value pair
  --
  -- If arg1 and arg2 are not nil, then arg1 is the key and arg2 is the
  -- value of a new table entry.
  --
  -- # Index value
  --
  -- If arg2 is nil, then arg1 is the value and is added as an indexed
  -- (by an integer) value.
  --
  -- @tparam table table
  -- @tparam mixed arg1
  -- @tparam mixed arg2
  --
  -- @treturn table
  local add_to_table = function(table, arg1, arg2)
    if arg2 == nil then
      local index = #table + 1
      return rawset(table, index, arg1)
    else
      return rawset(table, arg1, arg2)
    end
  end
```

```
182
183     return Pattern({
184       'list',
185
186       -- list_item*
187       list = CaptureFolding(
188         CaptureTable('') * Variable('list_item')^0,
189         add_to_table
190       ),
191
192       -- '{' list '}'
193       list_container =
194         ws('{') * Variable('list') * ws('}'),
195
196       -- ( list_container / key_value_pair / value ) ','?
197       list_item =
198         CaptureGroup(
199           Variable('list_container') +
200           Variable('key_value_pair') +
201           Variable('value')
202         ) * ws(',')^-1,
203
204       -- key '=' (list_container / value)
205       key_value_pair =
206         (Variable('key') * ws('=')) * (Variable('list_container') +
          ↪  Variable('value')),
207
208       -- number / string_quoted / string_unquoted
209       key =
210         Variable('number') +
211         Variable('string_quoted') +
212         Variable('string_unquoted'),
213
214       -- boolean !value / dimension !value / number !value / string_quoted !value /
          ↪  string_unquoted
215       -- !value -> Not-predicate -> * -Variable('value')
216       value =
217         Variable('boolean') * -Variable('value') +
218         Variable('dimension') * -Variable('value') +
219         Variable('number') * -Variable('value')  +
220         Variable('string_quoted') * -Variable('value') +
221         Variable('string_unquoted'),
222
223       -- boolean_true / boolean_false
224       boolean =
225         (
226           Variable('boolean_true') * CaptureConstant(true) +
227           Variable('boolean_false') * CaptureConstant(false)
228         ),
229
230       boolean_true =
231         Pattern('true') +
232         Pattern('TRUE') +
233         Pattern('True'),
234
235       boolean_false =
236         Pattern('false') +
237         Pattern('FALSE') +
238         Pattern('False'),
239
240       dimension = (
241         Variable('sign')^0 * white_space^0 *
```

```
      Variable('tex_number') * white_space^0 *
      Variable('unit')
    ) / capture_dimension,

    number =
      (white_space^0 * (Variable('lua_number') / tonumber) * white_space^0) ,

    tex_number =
      (Variable('integer')^1 * (Pattern('.') * Variable('integer')^1)^0) +
      (Pattern('.') * Variable('integer')^1),

    -- 'bp' / 'BP' / 'cc' / etc.
    -- https://raw.githubusercontent.com/latex3/lualibs/master/lualibs-util-dim.lua
    unit =
      Pattern('bp') + Pattern('BP') +
      Pattern('cc') + Pattern('CC') +
      Pattern('cm') + Pattern('CM') +
      Pattern('dd') + Pattern('DD') +
      Pattern('em') + Pattern('EM') +
      Pattern('ex') + Pattern('EX') +
      Pattern('in') + Pattern('IN') +
      Pattern('mm') + Pattern('MM') +
      Pattern('nc') + Pattern('NC') +
      Pattern('nd') + Pattern('ND') +
      Pattern('pc') + Pattern('PC') +
      Pattern('pt') + Pattern('PT') +
      Pattern('sp') + Pattern('SP'),

    lua_number =
      Variable('int') *
      Variable('frac')^-1,

    int = Variable('sign')^-1 * (
      Range('19') * Variable('integer') + Variable('integer')
    ),

    frac = Pattern('.') * Variable('integer'),
    sign = Set('-+'),
    integer = Range('09')^1,

    -- '"' ('\"' / !'"')* '"'
    string_quoted =
      white_space^0 * Pattern('"') *
      CaptureSimple((Pattern('\\"') + 1 - Pattern('"'))^0) *
      Pattern('"') * white_space^0,

    string_unquoted =
      white_space^0 *
      CaptureSimple(
        Variable('word_unquoted')^1 *
        (Set(' \t')^1 * Variable('word_unquoted')^1)^0) *
      white_space^0,

    word_unquoted = (1 - white_space - Set('{},='))^1
  })
end

--- Get the size of an array like table `{ 'one', 'two', 'three' }` = 3.
--
-- @tparam table value A table or any input.
--
-- @treturn number The size of the array like table. 0 if the input is
```

```lua
-- no table or the table is empty.
local function get_array_size(value)
  local count = 0
  if type(value) == 'table' then
    for _ in ipairs(value) do count = count + 1 end
  end
  return count
end

--- Get the size of a table `{ one = 'one', 'two', 'three' }` = 3.
--
-- @tparam table value A table or any input.
--
-- @treturn number The size of the array like table. 0 if the input is
-- no table or the table is empty.
local function get_table_size(value)
  local count = 0
  if type(value) == 'table' then
    for _ in pairs(value) do count = count + 1 end
  end
  return count
end

--- Unpack a single valued array table like `{ 'one' }` into `one` or
-- `{ 1 }` into `1`.
--
-- @treturn If the value is a array like table with one non table typed
-- value in it, the unpacked value, else the unchanged input.
local function unpack_single_valued_array_table(value, options)
  if
    type(value) == 'table' and
    get_array_size(value) == 1 and
    get_table_size(value) == 1 and
    type(value[1]) ~= 'table'
  then
    if type(value[1]) == 'string' and options.standalone_as_true then
      return value
    else
      return value[1]
    end
  end
  return value
end

local function visit_parse_tree(parse_tree, callback_func)
  if type(parse_tree) ~= 'table' then
    throw_error('Parse tree has to be a table')
  end
  local function visit_parse_tree_recursive(root_table, current_table, result,
  ↪  depth, callback_func)
    for key, value in pairs(current_table) do
      if type(value) == 'table' then
        value = visit_parse_tree_recursive(root_table, value, {}, depth + 1,
        ↪  callback_func)
      end

      key, value = callback_func(key, value, depth, current_table, root_table)

      if key ~= nil and value ~= nil then
        result[key] = value
      end
    end
```

```
364       if next(result) ~= nil then
365         return result
366       end
367     end
368
369     return visit_parse_tree_recursive(parse_tree, parse_tree, {}, 1, callback_func)
370   end
371
372   --- Normalize the result tables of the LPeg parser. This normalization
373   --  tasks are performed on the raw input table coming directly from the
374   --  PEG parser:
375   --
376   -- * Unpack all single valued array like tables: `{ 'text' }` into
377   --     `text`
378   --
379   -- @tparam table raw The raw input table coming directly from the PEG
380   --   parser
381   --
382   -- @tparam table options Some options. A table with the key
383   --    `unpack_single_array_values`
384   --
385   -- @treturn table A normalized table ready for the outside world.
386   local function normalize(raw, options)
387     local function normalize_recursive(raw, result, options)
388       for key, value in pairs(raw) do
389         if options.unpack_single_array_values then
390           value = unpack_single_valued_array_table(value, options)
391         end
392         if type(value) == 'table' then
393           result[key] = normalize_recursive(value, {}, options)
394         else
395           result[key] = value
396         end
397       end
398       return result
399     end
400     raw = normalize_recursive(raw, {}, options)
401
402     if options.standalone_as_true then
403       raw = visit_parse_tree(raw, function (key, value)
404         if type(key) == 'number' and type(value) == 'string' then
405           return value, true
406         end
407         return key, value
408       end)
409     end
410
411     if options.case_insensitive_keys then
412       raw = visit_parse_tree(raw, function (key, value)
413         if type(key) == 'string' then
414           return key:lower(), value
415         end
416         return key, value
417       end)
418     end
419
420     return raw
421   end
422
423   --- Parse a LaTeX/TeX style key-value string into a Lua table. With
424   -- this function you should be able to parse key-value strings like
425   -- this example:
```

```
--
--     show,
--     hide,
--     key with spaces = String without quotes,
--     string="String with double quotes: ,{}=",
--     dimension = 1cm,
--     number = -1.2,
--     list = {one,two,three},
--     key value list = {one=one,two=two,three=three},
--     nested key = {
--       nested key 2= {
--         key = value,
--       },
--     },
--
-- The string above results in this Lua table:
--
--     {
--       'show',
--       'hide',
--       ['key with spaces'] = 'String without quotes',
--       string = 'String with double quotes: ,{}=',
--       dimension = 1864679,
--       number = -1.2,
--       list = {'one', 'two', 'three'},
--       key value list = {
--         one = 'one',
--         three = 'three',
--         two = 'two'
--       },
--       ['nested key'] = {
--         ['nested key 2'] = {
--           key = 'value'
--         }
--       },
--     }
--
-- @tparam string kv_string A string in the TeX/LaTeX style key-value
--   format as described above.
--
-- @tparam table options A table containing
-- settings: `convert_dimensions`, `unpack_single_array_values`,
↪  `standalone_as_true`, `converter`
--
-- @treturn table A hopefully properly parsed table you can do
-- something useful with.
local function parse (kv_string, options)
  if kv_string == nil then
    return {}
  end
  options = normalize_parse_options(options)

  local parser = generate_parser(options)
  local parse_tree = parser:match(kv_string)

  if options.converter ~= nil and type(options.converter) == 'function' then
    parse_tree = visit_parse_tree(parse_tree, options.converter)
  end
  return normalize(parse_tree, options)
end

--- Convert back to strings
```

```lua
-- @section

--- The function `render(tbl)` reverses the function
--  `parse(kv_string)`. It takes a Lua table and converts this table
--  into a key-value string. The resulting string usually has a
--  different order as the input table. In Lua only tables with
--  1-based consecutive integer keys (a.k.a. array tables) can be
--  parsed in order.
--
-- @tparam table tbl A table to be converted into a key-value string.
--
-- @treturn string A key-value string that can be passed to a TeX
-- macro.
local function render (tbl)
  local function render_inner(tbl)
    local output = {}
    local function add(text)
      table.insert(output, text)
    end
    for key, value in pairs(tbl) do
      if (key and type(key) == 'string') then
        if (type(value) == 'table') then
          if (next(value)) then
            add(key .. '={')
            add(render_inner(value))
            add('},')
          else
            add(key .. '={},')
          end
        else
          add(key .. '=' .. tostring(value) .. ',')
        end
      else
        add(tostring(value) .. ',')
      end
    end
    return table.concat(output)
  end
  return render_inner(tbl)
end

--- The function `stringify(tbl, for_tex)` converts a Lua table into a
--   printable string. Stringify a table means to convert the table into
--   a string. This function is used to realize the `print` function.
--   `stringify(tbl, true)` (`for_tex = true`) generates a string which
--   can be embeded into TeX documents. The macro `\luakeysdebug{}` uses
--   this option. `stringify(tbl, false)` or `stringify(tbl)` generate a
--   string suitable for the terminal.
--
-- @tparam table input A table to stringify.
--
-- @tparam boolean for_tex Stringify the table into a text string that
--   can be embeded inside a TeX document via tex.print(). Curly braces
--   and whites spaces are escaped.
--
-- https://stackoverflow.com/a/54593224/10193818
local function stringify(input, for_tex)
  local line_break, start_bracket, end_bracket, indent

  if for_tex then
    line_break = '\\par'
    start_bracket = '$\\{$'
```

```
549       end_bracket = '$\\}$'
550       indent = '\\ \\ '
551     else
552       line_break = '\n'
553       start_bracket = '{'
554       end_bracket = '}'
555       indent = '  '
556     end
557
558     local function stringify_inner(input, depth)
559       local output = {}
560       depth = depth or 0
561
562       local function add(depth, text)
563         table.insert(output, string.rep(indent, depth) .. text)
564       end
565
566       local function format_key(key)
567         if (type(key) == 'number') then
568           return string.format('[%s]', key)
569         else
570           return string.format('[\'%s\']', key)
571         end
572       end
573
574       if type(input) ~= 'table' then
575         return tostring(input)
576       end
577
578       for key, value in pairs(input) do
579         if (key and type(key) == 'number' or type(key) == 'string') then
580           key = format_key(key)
581
582           if (type(value) == 'table') then
583             if (next(value)) then
584               add(depth, key .. ' = ' .. start_bracket)
585               add(0, stringify_inner(value, depth + 1))
586               add(depth, end_bracket .. ',');
587             else
588               add(depth, key .. ' = ' .. start_bracket .. end_bracket .. ',')
589             end
590           else
591             if (type(value) == 'string') then
592               value = string.format('\'%s\'', value)
593             else
594               value = tostring(value)
595             end
596
597             add(depth, key .. ' = ' .. value .. ',')
598           end
599         end
600       end
601
602       return table.concat(output, line_break)
603     end
604
605     return start_bracket .. line_break .. stringify_inner(input, 1) .. line_break ..
      ↪  end_bracket
606   end
607
608   --- The function `pretty_print(tbl)` pretty prints a Lua table to standard
609   --   output (stdout). It is a utility function that can be used to
```

```lua
--   debug and inspect the resulting Lua table of the function
--   `parse`. You have to compile your TeX document in a console to
--   see the terminal output.
--
-- @tparam table tbl A table to be printed to standard output for
-- debugging purposes.
local function pretty_print(tbl)
  print(stringify(tbl, false))
end

--- Store results
-- @section

--- A table to store parsed key-value results.
local result_store = {}

--- The function `save(identifier, result): void` saves a result (a
--  table from a previous run of `parse`) under an identifier.
--  Therefore, it is not necessary to pollute the global namespace to
--  store results for the later usage.
--
-- @tparam string identifier The identifier under which the result is
--   saved.
--
-- @tparam table result A result to be stored and that was created by
--   the key-value parser.
local function save(identifier, result)
  result_store[identifier] = result
end

--- The function `get(identifier): table` retrieves a saved result
--  from the result store.
--
-- @tparam string identifier The identifier under which the result was
--   saved.
local function get(identifier)
  -- if result_store[identifier] == nil then
  --   throw_error('No stored result was found for the identifier \'' .. identifier
  ↪  .. '\'')
  -- end
  return result_store[identifier]
end

--- Exports
-- @section

local export = {
  --- @see default_options
  default_options = default_options,

  --- @see stringify
  stringify = stringify,

  --- @see parse
  parse = parse,

  --- @see render
  render = render,

  --- @see pretty_print
  print = pretty_print,

```

```lua
    --- @see save
    save = save,

    --- @see get
    get = get,
  }

  -- http://olivinelabs.com/busted/#private
  if _TEST then
    export.luafy_key = luafy_key
    export.luafy_options = luafy_options
    export.normalize = normalize
    export.normalize_parse_options = normalize_parse_options
    export.unpack_single_valued_array_table = unpack_single_valued_array_table
    export.visit_parse_tree = visit_parse_tree
  end

  return export
```

6.2 luakeys.tex

```
%% luakeys.tex
%% Copyright 2021-2022 Josef Friedrich
%
% This work may be distributed and/or modified under the
% conditions of the LaTeX Project Public License, either version 1.3c
% of this license or (at your option) any later version.
% The latest version of this license is in
%   http://www.latex-project.org/lppl.txt
% and version 1.3c or later is part of all distributions of LaTeX
% version 2008/05/04 or later.
%
% This work has the LPPL maintenance status `maintained'.
%
% The Current Maintainer of this work is Josef Friedrich.
%
% This work consists of the files luakeys.lua, luakeys.sty, luakeys.tex
% luakeys-debug.sty and luakeys-debug.tex.

\directlua{luakeys = require('luakeys')}
```

6.3 luakeys.tex

```
%% luakeys.tex
%% Copyright 2021-2022 Josef Friedrich
%
% This work may be distributed and/or modified under the
% conditions of the LaTeX Project Public License, either version 1.3c
% of this license or (at your option) any later version.
% The latest version of this license is in
%   http://www.latex-project.org/lppl.txt
% and version 1.3c or later is part of all distributions of LaTeX
% version 2008/05/04 or later.
%
% This work has the LPPL maintenance status `maintained'.
%
% The Current Maintainer of this work is Josef Friedrich.
%
% This work consists of the files luakeys.lua, luakeys.sty, luakeys.tex
% luakeys-debug.sty and luakeys-debug.tex.

\directlua{luakeys = require('luakeys')}
```

6.4 luakeys-debug.tex

```
%% luakeys-debug.tex
%% Copyright 2021-2022 Josef Friedrich
%
% This work may be distributed and/or modified under the
% conditions of the LaTeX Project Public License, either version 1.3c
% of this license or (at your option) any later version.
% The latest version of this license is in
%   http://www.latex-project.org/lppl.txt
% and version 1.3c or later is part of all distributions of LaTeX
% version 2008/05/04 or later.
%
% This work has the LPPL maintenance status `maintained'.
%
% The Current Maintainer of this work is Josef Friedrich.
%
% This work consists of the files luakeys.lua, luakeys.sty, luakeys.tex
% luakeys-debug.sty and luakeys-debug.tex.

\directlua{
  luakeys = require('luakeys')
}

% https://tex.stackexchange.com/a/418401/42311
\catcode`\@=11
\long\def\LuaKeysIfNextChar#1#2#3{%
  \let\@tmpa=#1%
  \def\@tmpb{#2}%
  \def\@tmpc{#3}%
  \futurelet\@future\LuaKeysIfNextChar@i%
}%
\def\LuaKeysIfNextChar@i{%
  \ifx\@tmpa\@future%
    \expandafter\@tmpb
  \else
    \expandafter\@tmpc
  \fi
}%
\def\luakeysdebug@parse@options#1{
  \directlua{
    luakeys.save('debug_options', luakeys.parse('#1'))
  }
}%
\def\luakeysdebug@output#1{
  {
    \tt
    \parindent=0pt
    \directlua{
      local result = luakeys.parse('\luaescapestring{\unexpanded{#1}}',
      ↪  luakeys.get('debug_options'))
      tex.print(luakeys.stringify(result, true))
      luakeys.print(result)
    }
  }
}%
\def\luakeysdebug@oarg[#1]#2{%
  \luakeysdebug@parse@options{#1}%
  \luakeysdebug@output{#2}%
}%
\def\luakeysdebug@marg#1{%
  \luakeysdebug@output{#1}%
}%
```

```
\def\luakeysdebug{\LuaKeysIfNextChar[{\luakeysdebug@oarg}{\luakeysdebug@marg}}%
\catcode`\@=12
```

6.5 luakeys-debug.sty

```
%% luakeys-debug.sty
%% Copyright 2021-2022 Josef Friedrich
%
% This work may be distributed and/or modified under the
% conditions of the LaTeX Project Public License, either version 1.3c
% of this license or (at your option) any later version.
% The latest version of this license is in
%   http://www.latex-project.org/lppl.txt
% and version 1.3c or later is part of all distributions of LaTeX
% version 2008/05/04 or later.
%
% This work has the LPPL maintenance status `maintained'.
%
% The Current Maintainer of this work is Josef Friedrich.
%
% This work consists of the files luakeys.lua, luakeys.sty, luakeys.tex
% luakeys-debug.sty and luakeys-debug.tex.

\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{luakeys-debug}[2022/04/04 v0.5 Debug package for luakeys.]

\input luakeys-debug.tex
```

Change History