The luakeys package

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

v0.1 from 2021/01/18

```
local luakeys = require('luakeys')
local kv = luakeys.parse('level1={level2={level3={dim=1cm,bool=true,num=-1e-
↪  03,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 Syntax of the recognized key-value format

2.1 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}}`).

2.2 An (incomplete) attempt to put the syntax into the (Extended) Backus-Naur form

⟨*list*⟩ ::= ⟨*list-item*⟩ | ⟨*list-item*⟩ ⟨*list*⟩

⟨*list-item*⟩ ::= (⟨*key-value-pair*⟩ | ⟨*value-without-key*⟩) [',']

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

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

... to be continued

2.3 Recognized data types

2.3.1 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,
}
```

2.3.2 number

```
\luakeysdebug{
  num1 = 4,
  num2 = -4,
  num3 = 0.4,
  num4 = 4.57e-3,
  num5 = 0.3e12,
  num6 = 5e+20
}
```

```
{
  ['num1'] = 4,
  ['num2'] = -4,
  ['num3'] = 0.4,
  ['num4'] = 0.00457,
  ['num5'] = 300000000000.0,
  ['num6'] = 5e+20
}
```

2.3.3 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 used 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,
}
```

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

To learn more about the individual functions (local functions), pleasecommand 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
```

3.1 `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 a options table. This two options are supported.

```
local result = parse('one,two,three', {
  convert_dimensions = false,
  unpack_single_array_value = false
})
```

3.2 `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)
```

3.3 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 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',
}
```

4.1 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
```

4.2 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}
```

5 Implementation

5.1 luakeys.lua

```
1  -- luakeys-debug.tex
2  -- Copyright 2021 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-debug.sty
17 -- 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-lpeg.pdf)
30 --
31 -- @module luakeys
32
33 local lpeg = require('lpeg')
34
35 if not tex then
36   tex = {}
37
38   -- Dummy function for the tests.
39   tex['sp'] = function (input)
40     return 1234567
41   end
42 end
43
44 --- Generate the PEG parser using Lpeg.
45 --
46 -- @treturn userdata The parser
47 local function generate_parser(options)
48   -- Optional whitespace
49   local white_space = lpeg.S(' \t\n\r')^0
50
51   --- Match literal string surrounded by whitespace
52   local ws = function(match)
```

```
    return white_space * lpeg.P(match) * white_space
  end

  local boolean_true =
    lpeg.P('true') +
    lpeg.P('TRUE') +
    lpeg.P('True')

  local boolean_false =
    lpeg.P('false') +
    lpeg.P('FALSE') +
    lpeg.P('False')

  local number = lpeg.P({'number',
    number =
      lpeg.V('int') *
      lpeg.V('frac')^-1 *
      lpeg.V('exp')^-1,

    int = lpeg.V('sign')^-1 * (
      lpeg.R('19') * lpeg.V('digits') + lpeg.V('digit')
    ),

    sign = lpeg.S('+-'),
    digit = lpeg.R('09'),
    digits = lpeg.V('digit') * lpeg.V('digits') + lpeg.V('digit'),
    frac = lpeg.P('.') * lpeg.V('digits'),
    exp = lpeg.S('eE') * lpeg.V('sign')^-1 * lpeg.V('digits'),
  })

  --- Define data type dimension.
  --
  -- @return Lpeg patterns
  local function build_dimension_pattern()
    local sign = lpeg.S('-+')
    local integer = lpeg.R('09')^1
    local tex_number = (integer^1 * (lpeg.P('.') * integer^1)^0) + (lpeg.P('.') *
    ↪  integer^1)
    local unit
    -- https://raw.githubusercontent.com/latex3/lualibs/master/lualibs-util-dim.lua
    for _, dimension_extension in ipairs({'bp', 'cc', 'cm', 'dd', 'em', 'ex', 'in',
    ↪  'mm', 'nc', 'nd', 'pc', 'pt', 'sp'}) do
      if unit then
        unit = unit + lpeg.P(dimension_extension)
      else
        unit = lpeg.P(dimension_extension)
      end
    end

    local dimension = (sign^0 * tex_number * unit)

    if options.convert_dimensions then
      return dimension / tex.sp
    else
      return lpeg.C(dimension)
    end
  end
```

```
108
109   --- Add values to a table in a two modes:
110   --
111   -- # Key value pair
112   --
113   -- If arg1 and arg2 are not nil, then arg1 is the key and arg2 is the
114   -- value of a new table entry.
115   --
116   -- # Index value
117   --
118   -- If arg2 is nil, then arg1 is the value and is added as an indexed
119   -- (by an integer) value.
120   --
121   -- @tparam table table
122   -- @tparam mixed arg1
123   -- @tparam mixed arg2
124   --
125   -- @treturn table
126   local add_to_table = function(table, arg1, arg2)
127     if arg2 == nil then
128       local index = #table + 1
129       return rawset(table, index, arg1)
130     else
131       return rawset(table, arg1, arg2)
132     end
133   end
134
135   return lpeg.P({
136     'list',
137
138     list = lpeg.Cf(
139       lpeg.Ct('') * lpeg.V('list_item')^0,
140       add_to_table
141     ),
142
143     list_container =
144       ws('{') * lpeg.V('list') * ws('}'),
145
146     list_item =
147       lpeg.Cg(
148         lpeg.V('key_value_pair') +
149         lpeg.V('value')
150       ) * ws(',')^-1,
151
152     key_value_pair =
153       (lpeg.V('key') * ws('=')) * (lpeg.V('list_container') + lpeg.V('value')),
154
155     -- ./ for tikz style keys
156     key_word = lpeg.R('az', 'AZ', '09', './'),
157
158     key = white_space * lpeg.C(
159       lpeg.V('key_word')^1 *
160       (lpeg.P(' ')^1 * lpeg.V('key_word')^1)^0
161     ) * white_space,
162
163     value =
164       lpeg.V('boolean') +
```

```
      lpeg.V('dimension') +
      lpeg.V('number') +
      lpeg.V('string_quoted') +
      lpeg.V('string_unquoted'),

    boolean =
      boolean_true * lpeg.Cc(true) +
      boolean_false * lpeg.Cc(false),

    dimension = build_dimension_pattern(),

    string_quoted =
      white_space * lpeg.P('"') *
      lpeg.C((lpeg.P('\\"') + 1 - lpeg.P('"'))^0) *
      lpeg.P('"') * white_space,

    string_unquoted =
      white_space *
      lpeg.C((1 - lpeg.S('{},='))^1) *
      white_space,

    number =
      white_space * (number / tonumber) * white_space,

  })
end

local function trim(input_string)
  return input_string:gsub('^%s*(.-)%s*$', '%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
-- 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
```

```lua
end

--- Unpack a single valued array table like `{ 'one' }` into `one` or
-- `{ 1 }` into `into`.
--
-- @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)
  if
    type(value) == 'table' and
    get_array_size(value) == 1 and
    get_table_size(value) == 1 and
    type(value[1]) ~= 'table'
  then
    return value[1]
  end
  return value
end

--- This normalization tasks are performed on the raw input table
-- coming directly from the PEG parser:
--
-- 1. Trim all strings: ` text \n` into `text`
-- 2. Unpack all single valued array like tables: `{ 'text' }`
--    into `text`
--
-- @tparam table raw The raw input table coming directly from the PEG
--   parser
--
-- @treturn table A normalized table ready for the outside world.
local function normalize(raw, options)
  local function normalize_recursive(raw, result, options)
    for key, value in pairs(raw) do
      if options.unpack_single_array_values then
        value = unpack_single_valued_array_table(value)
      end
      if type(value) == 'table' then
        result[key] = normalize_recursive(value, {}, options)
      elseif type(value) == 'string' then
        result[key] = trim(value)
      else
        result[key] = value
      end
    end
    return result
  end
  return normalize_recursive(raw, {}, options)
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.
--
```

```lua
-- @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 = '\\{'
    end_bracket = '\\}'
    indent = '\\ \\ '
  else
    line_break = '\n'
    start_bracket = '{'
    end_bracket = '}'
    indent = '  '
  end

  local function stringify_inner(input, depth)
    local output = {}
    depth = depth or 0;

    local function add(depth, text)
      table.insert(output, string.rep(indent, depth) .. text)
    end

    for key, value in pairs(input) do
      if (key and type(key) == 'number' or type(key) == 'string') then
        key = string.format('[\'%s\']', key);

        if (type(value) == 'table') then
          if (next(value)) then
            add(depth, key .. ' = ' .. start_bracket);
            add(0, stringify_inner(value, depth + 1, for_tex));
            add(depth, end_bracket .. ',');
          else
            add(depth, key .. ' = ' .. start_bracket .. end_bracket .. ',');
          end
        else
          if (type(value) == 'string') then
            value = string.format('\'%s\'', value);
          else
            value = tostring(value);
          end

          add(depth, key .. ' = ' .. value .. ',');
        end
      end
    end

    return table.concat(output, line_break)
  end

```

```
336     return start_bracket .. line_break .. stringify_inner(input, 1) .. line_break ..
        ↪  end_bracket
337   end
338
339   --- For the LaTeX version of the macro
340   --  `\luakeysdebug[options]{kv-string}`.
341   --
342   -- @tparam table options_raw Options in a raw format. The table may be
343   -- empty or some keys are not set.
344   --
345   -- @treturn table
346   local function normalize_parse_options (options_raw)
347     if options_raw == nil then
348       options_raw = {}
349     end
350     local options = {}
351
352     if options_raw['unpack single array values'] ~= nil then
353       options['unpack_single_array_values'] = options_raw['unpack single array
          ↪  values']
354     end
355
356     if options_raw['convert dimensions'] ~= nil then
357       options['convert_dimensions'] = options_raw['convert dimensions']
358     end
359
360     if options.convert_dimensions == nil then
361       options.convert_dimensions = true
362     end
363
364     if options.unpack_single_array_values == nil then
365       options.unpack_single_array_values = true
366     end
367
368     return options
369   end
370
371   return {
372     stringify = stringify,
373
374     --- Parse a LaTeX/TeX style key-value string into a Lua table. With
375     -- this function you should be able to parse key-value strings like
376     -- this example:
377     --
378     --     show,
379     --     hide,
380     --     key with spaces = String without quotes,
381     --     string="String with double quotes: ,{}=",
382     --     dimension = 1cm,
383     --     number = -1.2,
384     --     list = {one,two,three},
385     --     key value list = {one=one,two=two,three=three},
386     --     nested key = {
387     --       nested key 2= {
388     --         key = value,
389     --       },
390     --     },
```

```
391     --
392     -- The string above results in this Lua table:
393     --
394     --     {
395     --       'show',
396     --       'hide',
397     --       ['key with spaces'] = 'String without quotes',
398     --       string = 'String with double quotes: ,{}=',
399     --       dimension = 1864679,
400     --       number = -1.2,
401     --       list = {'one', 'two', 'three'},
402     --       key value list = {
403     --         one = 'one',
404     --         three = 'three',
405     --         two = 'two'
406     --       },
407     --       ['nested key'] = {
408     --         ['nested key 2'] = {
409     --           key = 'value'
410     --         }
411     --       },
412     --     }
413     --
414     -- @tparam string kv_string A string in the TeX/LaTeX style key-value
415     --   format as described above.
416     --
417     -- @tparam table options A table containing
418     -- settings: `convert_dimensions` `unpack_single_array_values`
419     --
420     -- @treturn table A hopefully properly parsed table you can do
421     -- something useful with.
422     parse = function (kv_string, options)
423       if kv_string == nil then
424         return {}
425       end
426       options = normalize_parse_options()
427
428       local parser = generate_parser(options)
429       return normalize(parser:match(kv_string), options)
430     end,
431
432     --- The function `render(tbl)` reverses the function
433     --  `parse(kv_string)`. It takes a Lua table and converts this table
434     --  into a key-value string. The resulting string usually has a
435     --  different order as the input table. In Lua only tables with
436     --  1-based consecutive integer keys (a.k.a. array tables) can be
437     --  parsed in order.
438     --
439     -- @tparam table tbl A table to be converted into a key-value string.
440     --
441     -- @treturn string A key-value string that can be passed to a TeX
442     -- macro.
443     render = function (tbl)
444       local function render_inner(tbl)
445         local output = {}
446         local function add(text)
447           table.insert(output, text)
```

```
448         end
449         for key, value in pairs(tbl) do
450           if (key and type(key) == 'string') then
451             if (type(value) == 'table') then
452               if (next(value)) then
453                 add(key .. '={');
454                 add(render_inner(value));
455                 add('},');
456               else
457                 add(key .. '={},');
458               end
459             else
460               add(key .. '=' .. tostring(value) .. ',');
461             end
462           else
463             add(tostring(value) .. ',')
464           end
465         end
466         return table.concat(output)
467       end
468       return render_inner(tbl)
469     end,
470
471     --- The function `print(tbl)` pretty prints a Lua table to standard
472     --   output (stdout). It is a utility function that can be used to
473     --   debug and inspect the resulting Lua table of the function
474     --   `parse`. You have to compile your TeX document in a console to
475     --   see the terminal output.
476     --
477     -- @tparam table tbl A table to be printed to standard output for
478     -- debugging purposes.
479     print = function(tbl)
480       print(stringify(tbl, false))
481     end,
482
483   }
```

5.2 luakeys-debug.tex

```
%% luakeys-debug.tex
%% Copyright 2021 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-debug.sty
% and luakeys-debug.tex.

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

\def\luakeysdebug#1{
  {
    \tt
    \parindent=0pt
    \directlua{
      local result = luakeys.parse('#1')
      tex.print(luakeys.stringify(result, true))
      luakeys.print(result)
    }
  }
}
```

5.3 luakeys-debug.sty

```
%% luakeys-debug.sty
%% Copyright 2021 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-debug.sty
% and luakeys-debug.tex.

\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{luakeys-debug}[2021/01/18 Debug package for luakeys.]

%\input luakeys-debug.tex

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

\newcommand{\luakeysdebug}[2][]{
  {
    \tt
    \parindent=0pt
    \directlua{
      local options_raw = luakeys.parse('#1')
      local result = luakeys.parse('#2', options_raw)
      tex.print(luakeys.stringify(result, true))
      luakeys.print(result)
    }
  }
}
```

Change History

v0.1

General: Inital release 21