

# luatbls

Create, modify, and use Lua tables from within LaTeX

Kale Ewasiuk (kalekje@gmail.com)

2025-02-14

## Introduction

This package provides a Lua-table interface based on the luakeys package:

<https://mirror.quantum5.ca/CTAN/macros/luatex/generic/luakeys/luakeys.pdf>

A global table called `luatbls` is created by loading this package. This table contains all user-defined tables as well as internal package functions and settings. User tables are stored directly under the module's table, so you can access a table within Lua by using: `luatbls['mytable']` or `luatbls.mytable`. Further, `luatbls` can be called directly to obtain a table item by `luatbls'i'`, where `i` is a “flexible” indexing system discussed in the next paragraphs.

If you want to change the luakeys global parser options, you can adjust them by:

```
\directlua {luatbls._luakeys.opts.OPTION = VALUE}
```

For debugging, set `\directlua {luatbls._debug = true}`

In this documentation, arguments are represented as follows:

`t` : table name. If none provided, the most recent is used.

`k` : a string key.

`n` : an integer index.

`v` : a value.

`i` : the flexible indexer to get a single item.

`I` : the flexible indexer to get a single or multiple items.

`keyval` : a key-value string for the table. Standalone values are set to boolean.

`csv` : a key-value string where standalone values are parsed as array-like.

`opts` : options for `luakeys.parse()`.

`cstemp` : a template for command-sequences, lengths, or toggles. By default is `1tbl<t><k>`.

There are a few ways to use the index (placeholder `i`).

`t.k` where `t` is the table name and `k` is a string key (i.e. uses `luatbls.t.k`),

`t/n` where `n` is an integer index (i.e. uses `t.k[n]`); note that negative indexes are allowed where `-1` is the last element. Alternatively, `t` and the symbol can be omitted, and simply pass the element without the table name as a prefix, where the assumed table is the last one that was created or changed to (i.e. the most 'recent' table). In this case, passing a number will assume an integer index.

To use a `I`, you can select tables and groups of keys by `t|seq`, or `t.k`, or `t/n`. If no `|./` is provided, the recent table is used and the argument is assumed to be a sequence of keys. `penlightplus`'s command `penlight.seq.tbltrain()` syntax is used for sequences. To summarize what `seq` can be, a comma-separated list of numbers or keys are used to specify which elements are iterated over. NumPy-like

slicing is possibly with : to choose integer ranges. If \* is provided, all string keys are iterated. If I is entirely blank, all elements of the recent table are used, which is equivalent to t|\*, :.

The cstemp default can be changed with: \luadirect {luatbls.\_cstemp = 'ltbl<t><k>'}, where <t> and <k> are the table and key names. Numerical keys are converted to capital letters: 1->A, 2->B. It is recommended that tables and keys contain letters only for predictable behaviour when using this feature. If the value of a tbl's key is a table, every element in that table is defined, and the keys of that nested table is appended to the cs: ltbl<t><k1><k2> (noting that numbers are converted to letters).

Note: nested tables are currently not fully supported. Some variations of commands have an E suffix which indicates that nested elements can be explicitly indexed. The table name must be specified, and the validity of table names and keys are not checked.

The tbl commands fully expand the t, k, n, i, and I arguments. However a variation with an N-appended is usually provided which will not expand the v, keyval, or csv args.

## Creating Tables

\tblnew {t} declares a new table with name t

\tblchg {t} changes the 'recent' table

\tblfrkv {t}{keyval}[opts] new table from key-vals using luakeys

\tblfrkvN {t}{keyval}[opts] does not expand key-val string luakeys. Note: opts are parsed using luakeys with the naked\_as\_value=true, so booleans must be explicitly set.

\tblfrcsv {t}{csv}[opts] a shorthand \tblfrkv {t}{csv}[naked\_as\_value=true,opts], a good way to convert a comma-separated list to an array

\tblfrcsvN {t}{csv}[opts] same as above, but the csv is not expanded.

\tblkvundefcheck will throw an error if you use define a table from key-values and use a key that was not specified in the luakeys parse options via opts.defaults or opts.defs.

\tblenforcechoices {i}{csv} will throw an error if the value of item i is not in the provided csv

## Setting, getting, and modifying

\tblset {i}{v} sets a value of the table/index i to v

\tblsetN {i}{v} same as above, but the value is not expanded.

\tblget {i} gets the value and tex.sprint()s it

\tblgetE {t.k} An 'explicit' version of tbl get. Use this for nested tables. The tbl name must be specified. The validity of table names and keys are not checked.

\tblsetE {i}{v} the explicit version of \tblset . Quotes must be used for strings in the v, and arbitrary lua code can be entered.

```

1 \tblfrkv{ex}{a,b,c,first=john,last=←
    smith}%
2 [defaults={x=0,1=one,n=false,y=yes←
    }]
3 \tblget{ex.a}\\ true
4 \tblset{a}{tRuE!!} tRuE!!
5 \tblget{a}\\ 0
6 \tblget{ex.x}\\ 0
7 \tblget{.x}\\ val
8 \tbladd{ex.newkey}{val}\tblget{newkey← VAL
    }\\ 1
9 \tbladd{nk}{VAL}\tblget{nk}\\
10 \tblsetE{ex.d}{math.mod2(3)}
11 \tblget{d}

1 \tblfrcsv{EX}{x={1,2,{1,2,3}},name=me} 1
2 \tblgetE{EX.x[1]}\\
3 \tblsetE{EX.x[3][3]}{99}\\ 99
4 \tblgetE{EX.x[3][3]}\\ me
5 \tblgetE{EX.name}\\

```

`\tbladd {i}{v}` add a new value to a table using index method  
`\tbladdN {i}{v}` above, but don't expand the value argument

`\tblapp {t}{v}` append a value (integer-wise) to a table  
`\tblappN {t}{v}`

`\tblupd {t}{keyval}` update a table with more keyvals  
`\tblupdN {t}{keyval}`

`\tblcon {t}{csv}` concatenate array-style `csv` at the end of `t`  
`\tblconN {t}{csv}`

## Conditionals

`\tblif {i}{tr}[fa]` runs code `tr` if the item is true else `fa`  
`\tblifv {i}{tr}[fa]` runs code `tr` if the item is truth-y (using `pl.hasval`) else `fa`  
`\tblifeq {i}{v}{tr}[fa]` checks the equivalency of `i` to a user-specified value. The value is fully expanded. Quotes must be used to indicate strings.  
`\tblifeqstr {i}{v}{tr}[fa]` checks the equivalency of `i` to a user-specified value to a string (uses `luastr`).

```

1 \def\JJ{1}
2 \tblfrcsv{x}{n=false,y=true,
3 k0="",kv=val,k2=6,
4 k1=1,k11="1",
5 }
6 \tblif{n}{tr}[FA]\\ FA
7 \tblif{k0}{TR}[fa]\\ TR
8 \tblifv{k0}{tr}[FA]\\ TR
9 \tblifeq{kv}{'val'}{TR}[fa]\\ Tr
10 \tblifeq{k2}{6}{TR}[fa]\\
11 \tblifeq{k1}{\JJ}{Tr}
12 \tblifeqstr{k11}{\JJ}{Tr}

```

## Iterating

`\tblfor {I}{template}` and `\tblforN` By default, iterates over all elements (`seq = *, :`), but arbitrary indices/keys can be iterated over as per `penlight.seq.tbltrain` syntax. `<k>` and `<v>` are placeholders in the template that are replaced by the keys and vals and can be changed by:

`\luadirect {luaTbls._tblv = '<v>'}`

If you want to iterate over a second-level table, you must use:

`\tblforE` and `\tblforEN`, and explicitly provide the table and element.

```

1 \tblfrcsv{x}{n1,k1=v1,n2,n3,n4,
2   k2=v2,k3=v3,n5,n6}
3 1> \tblfor{:}{<k> = <v>; }\\
4 2> \tblfor*}{<k> = <v>; }\\
5 3> \tblfor{1,*,2::2}{<k> = <v>; }\\
6 4> \tblfor{ x | 1,*,2::2}{<k> = <v>; }\\
7 \tblfrcsv{x}{a,{a,b,c}}
8 5> \tblforE{x[2]}{<k> = <v>; }
```

1> 1 = n1; 2 = n2; 3 = n3; 4 = n4; 5 = n5; 6 = n6;  
 2> k1 = v1; k2 = v2; k3 = v3;  
 3> 1 = n1; k1 = v1; k2 = v2; k3 = v3; 2 = n2; 4 = n4; 6 = n6;  
 4> 1 = n1; k1 = v1; k2 = v2; k3 = v3; 2 = n2; 4 = n4; 6 = n6;  
 5> 1 = a; 2 = b; 3 = c;

## Definitions

`\tbldef {i}[cstemp]` pushes the value to macro `cstemp`.

`\tblgdef {i}[cstemp]` like above but global definition is used.

`\tbldefs {I}[cstemp]` and `\tblgdefs {I}[cstemp]` defines items in table `t` (use recent if blank).

```

1 \tblfrcsv{EX}{n1,kA=v1,n2,n3,n4,
2   kB=v2,kC=v3,n5,n6}
3 1>\tbldef{kA}[mycs]\mycs\tbldef{kA}\
   \tblEXkA\\
4 2> \tbldef{EX/1}\tblEXA
```

1>v1v1  
 2> n1

```

1 \tblfrcsv{EX}{x={1,2,3}}
2 1>\tbldef{x}[mycs]\mycsA, \mycsB \\
3 2>\tbldefs{}\tblEXxA, \tblEXxB
```

1>1, 2  
 2>1, 2

`\tbldefxy {i}[cstemp]` splits the value of item by space, and creates two definitions `<cstemp>x` and `<cstemp>y`. This might be useful for passing and using tikz coordinates, for example `xy=0 5`. An error is thrown if the values are non-numeric.

```

1 \tblfrkv{EX}{coords=12 34,other}
2 \tbldefxy{coords}[d]\dx, \dy \\
3 \tbldefxy{coords}\tblEXcoordsx, \tblEXcoordsy \\
```

12, 34  
 12, 34

`\tblmaketoggle {i}[cstemp]` will create and set a toggle (see `etoolbox`) for a truth-y value (see `p1.hasval`)

`\tblmaketoggles {I}[cstemp]` will iterate over `I` and create and set global toggles (see `etoolbox`) for boolean values

```

1 \tblfrkv{ex}{atog=true,!btog} True
2 \tblmakegtoggles{} False
3 \iftoggle{tbllexatog}{True}{}\
4 \iftoggle{tbllexbtog}{}{False}\

```

`\tblmakelength {i}[cstemp]` will 'forcefully' create a length for an element. Glue expressions are permitted. See `etoolbox's \deflength {}`

`\tblmakelengths {I}[cstemp]` will iterate over `I` and create global lengths for elements that are tex dimensions. If plain numbers are found, `sp` units are used (in case the `convert_dimensions=true` `luakeys` option is used, which converts to `sp`)

```

1 \tblfrkv{ex}{alen=1cm,blen=2cm,↵
   clen=10mm*2+2cm}[convert_↵
   dimensions=true]
2 \tblmakelengths{}[<k>]
3 I\hspace{\alen}I\
4 I\hspace{\blen}I\
5 \tblmakelength{clen}[LEN]
6 I\hspace{\LEN}I

```

## Utilities

`\tblapply {I}{func1(<v>,x,y)|:func2}[newtable]` apply a Lua function(s).

If `newtable` is provided, a new table is created (and made the recent table) and the original table is preserved.

The `.`, `/` or `|` indexer may be used to apply a function to a single value or group of keys. Multiple functions can be applied sequentially, separated by `|`.

An arbitrary global function (including additional arguments) can be used, but if a function is prefixed with a `:`, the class method will be called. The `stringx` and `tablex` methods from `penlight` are used depending on the value's type. See:

<https://lunarmodules.github.io/Penlight/>

Arguments can be specified with round brackets, where `<v>` and `<k>` are used as a placeholder for the values and keys. If no arguments are passed, it is assumed that the value is the only argument. Note that `luakeys` parses the args, so quotes are not needed around strings for the args.

```

1 \tblfrcsv{ex}{{a, b, c}}
2 \tblapply{}{:concat(<v>,-) | :upper}[↵
   new]
3 1> \tblgetE{ex}[1][1]\
4 2> \tblget{new/1}\
5 \tblfrcsv{ex}{HelloWorld}
6 \tblapply{}{string.sub(<v>,2,-5)}[new]
7 3> \tblget{new/1}

```

```

1> a
2> A-B-C
3> elloW

```

`\tblprt {t}` pretty-print the table in console. Using `\tblprt *{}` will terminate the LaTeX program immediately after and issue an error, which could be useful for debugging.

## An Example

```

1 \NewDocumentCommand{\Exampletbl}{m}{
2   \tblfrcsv{ex}{#1}[defaults={sal=
3     Hello}]
4   %\tblkvundefcheck
5   \tblapply{ex.auth}{:list2comma}
6   \tblget{sal}, \tblget{auth}! Thank
7     you for writing such a great
8     novel.
9   My favorite parts were:
10  \begin{description}
11    \tblforEN{ex.chaps}{\item[<k>]
12      <v> }
13  \end{description}
14  It was also very cool to learn
15    that
16    \tblgetE{ex.num[1]}*\tblgetE{ex.
17      num[2]}=
18    \luadirect{tex.sprint(tostring(
19      luatbls.ex.num[1]*luatbls.ex.
20      num[2]))}
21 }
22 \Exampletbl{auth={You,Me,Dupree},
23   chaps={intro=very enticing, climax
24     =thrilling, finale=what a
25     twist!}
26   num={12,13}
27 }

```

Hello, You, Me, and Dupree! Thank you for writing such a great novel. My favorite parts were:

**climax** thrilling

**finale** what a twist!

**intro** very enticing

It was also very cool to learn that  $12 \cdot 13 = 156$