penlightplus
Additions to the Penlight Lua Libraries

Kale Ewasiuk (kalekje@gmail.com)

2024–03–14

This package first loads the [import]penlight package. The pl option may be passed to this package to create an alias for penlight. The globals option may be used to make several of the functions global (as discussed below).

texlua usage

If you want to use penlightplus.lua with the texlua interpreter (no document is made, but useful for testing your Lua code), you can access it by setting __SKIP_TEX__ = true before loading. For example:

```lua
package.path = package.path .. ';'..'path/to/texmf/tex/lualatex/penlightplus/?.lua'
package.path = package.path .. ';'..'path/to/texmf/tex/lualatex/penlight/?.lua'
penlight = require'penlight'
__SKIP_TEX__ = true --only required if you want to use penlightplus without a LaTeX run
__PL_GLOBALS__ = true -- optional, include global definitions
require'penlightplus'
```

The following global Lua variables are defined:

- __SKIP_TEX__ If using the penlightplus package with texlua (good for troubleshooting), set this global before loading penlight
- The goals flags below are taken care of in the package options:
- __PL_GLOBALS__ If using package with texlua and you don’t want to set some globals (described in next sections), set this global before to true loading penlight
__PL_NO_HYPERREF__ a flag used to change the behaviour of a function, depending on if you don’t use the hyperref package

__PDFmetadata__ a table used to store PDF meta-data

**global extras**

If the package option `globals` is used, many additional globals are set for easier scripting. `pl.hasval`, `pl.COMP`, `pl.utils.kpairs`, `pl.utils.npairs` become globals. `pl.tablex` is aliased as `pl.tbx` and `tbx` (which also includes all native Lua table functions), and `pl.array2d` is aliased as `pl.a2d` and `a2d`.

If you want global `pl.tex` funcs and vars, call `pl.make_tex_global()`

**penlight additions**

Some functionality is added to penlight and Lua.

`pl.hasval(x)` Python-like boolean testing

`COMP'xyz'()` Python-like comprehensions:

https://lunarmodules.github.io/Penlight/libraries/pl.comprehension.html

`clone_function(f)` returns a cloned function

`operator.strgt(a,b)` compares strings a greater than b (useful for sorting)

`operator.strlt(a,b)` compares strings a less than b (useful for sorting)

`math.mod(n,d)`, `math.mod2(n)` math modulous

`string.upfirst(s)` uppercase first letter

`string.delspace(s)` delete all spaces

`string.trimfl(s)` remove first and last chars

`string.appif(s, append, bool, alternate)`

`string.gfirst(s, t)` return first matched pattern from an array of patterns t

`string.gextract(s)` extract a pattern from a string (returns capture and new string with capture removed)

`string.totable(s)` string a table of characters

`string.tolist(s)` string a table of characters

`string.containsany(s,t)` checks if any of the array of strings t are in s using `string.find`

`string.containsanycase(s,t)` case-insensitive version

`string.delspace(s)` clear spaces from string

`string.subpar(s, c)` replaces `\par` with a character of your choice default is space

`string.fmt(s, t, fmt)` format a string like `format_operator`, but with a few improvements.
t can be an array (reference items like \$1 in the string), and fmt can be a table of formats (keys correspond to those in t), or a string that is processed by luakeys.

string.parsekv(s, opts) parse a string using penlight.luakeys. A string or table can be used for opts.

tablex.fmt(t, f) format a table with table or key-value string f

tablex.strinds(t) convert integer indexes to string indices (1 -> '1')

tablex.filterstr(t,e,case) keep only values in table t that contain expression e, case insensitive by default.

tablex.mapslice(f,t,i1,i2) map a function to elements between i1 and i2

tablex.listcontains(t,v) checks if a value is in a array-style list

pl.char(n) return letter corresponding to 1=a, 2=b, etc.
pl.Char(n) return letter corresponding to 1=A, 2=B, etc.

pl.utils.filterfiles(dir,filt,rec) Get files from dir and apply glob-like filters. Set rec to true to include subdirectories

A pl.tex. module is added

add_bkt_cnt(n), close_bkt_cnt(n), reset_bkt_cnt functions to keep track of adding curly brackets as strings. add will return n (default 1) {'s and increment a counter. close will return n }'s (default will close all brackets) and decrement.
_NumBkts internal integer for tracking the number of brackets
opencmd(cs) prints \cs {} and adds to the bracket counters.

xNoValue,xTrue,xFalse: xparse equivalents for commands

prt(x),prtn(x) print without or with a newline at end. Tries to help with special characters or numbers printing.
prt1(1),prtt(t) print a literal string, or table
wrt(x), wrtn(x) write to log
    wrh(s1, s2) pretty-print something to console. S2 is a flag to help you find., alias is help_wrt, also in pl.wrth
prt_array2d(tt) pretty print a 2d array

pkgwarn(pkg, msg1, msg2) throw a package warning
pkgerror(pkg, msg1, msg2, stop) throw a package error. If stop is true, immediately ceases compile.

defcmd(cs, val) like \gdef , but note that no special chars allowed in cs(eg. @)
defmacro(cs, val) like \gdef, allows special characters, but any tokens in val must be predefined (this uses token.set_macro internally)

newcmd(cs, val) like \newcommand

renewcmd(cs, val) like \renewcommand

prvcmd(cs, val) like \providecommand

decmd(cs, dft, overwrite) declare a command. If dft (default) is nil, cs is set to a package warning saying 'cs' was declared and used in document, but never set. If overwrite is true, it will overwrite an existing command (using defcmd), otherwise, it will throw error like newcmd.

get_ref_info(l) accesses the \r @label and returns a table

Recording latex input

penlight.tex.startrecording() start recording input buffer without printing to latex
penlight.tex.stoprecording() stop recording input buffer
penlight.tex.readbuf() internal-use function that interprets the buffer. This will ignore an environment ending (eg. end{envir})

penlight.tex.recordedbuf the string variable where the recorded buffer is stored

Macro helpers

\MakeluastringCommands [def]{spec} will let \plluastring (A|B|C..) be \luastring (N|O|T|F) based on the letters that spec is set to (or def if nothing is provided) This is useful if you want to write a command with flexibility on argument expansion. The user can specify n, o, t, and f (case insensitive) if they want no, once, twice, or full expansion.
For example, we can control the expansion of args 2 and 3 with arg 1:

\NewDocumentCommand{\splittocomma}{ O{nn} m m }{%
  \MakeluastringCommands[nn]{#1}%
  \luadirect{penlight.tex.split2comma(\plluastringA{#2},\plluastringB{#3})}%
}

Lua boolean expressions for LaTeX conditionals

\ifluax {<Lua expr>}{<do if true>}{<do if false>} and
\ifluax {<Lua expr>}{<do if true>}{<do if false>} for truthy (uses penlight.hasval)
Creating and using Lua tables in LaTeX

penlightplus provides a Lua-table interface. Tables are stored in the penlight.tbls table.

\tblnew {t} declares a new table with name t
\tblchg {t} changes the 'recent' table
\tblfrkv {t}{{key-val string}[luakeys opts]} new table from key-vals using luakeys
\tblfrkvN {t}{{key-val string}[luakeys opts]} does not expand key-val string luakeys
\tblfrkvCD {t}{{key-val string}[luakeys opts]} define tbl from key-val, check if any were not defined as defaults (see below), and then push all to definitions
\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. \tblfrcsv {t}{csv} 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} same as above, but the csv is not expanded. \tblset {t}{i}{v} sets a value of the table/index i to v
\tblget {t}{i} gets the value and \tex.sprint()s it
\tbladd {t}{i}{v} add a new value to a table using index method
\tbladdN {t}{i}{v} above, but don’t expand the value argument
\tblconcat {t}{csv} concatenate an array-style csv
\tblappend {t}{v} append a value (integer-wise) to a table
\tbldef {t}{i}{d} pushes the value to macro d
\tbldefall {t}{{d}} define all item in table t (use recent if blank) with format d<key> where d is your prefix. If d is blank, keys will be defined as \dtbl <t><k> \tblgdef {i}{{d}} pushes the defined value to a global
\tbldefxy {t}{{d}} splits the value of item by spaces creates two definitions \dx and \dy. Useful for passing tikz coordinates like xy=0 5
For defining tables, if d is blank, commands are defined as \dtbl<t><k>
\iftbl {t}{i}{tr}{fa} runs code ta if the item is true else \ftblv {t}{i}{tr}{fa} runs code ta if the item is truthy else \ft
\tblprint {t} print the table in console

There are 3 ways to use the index (placeholder {i} above). t.key where t is the table name and key is a string key, t/int where int is an integer index (ie. uses t[int],
Note that negative indexes are allowed where -1 is the last element, or simply use \ind without the table name, where the assumed table is the last one that was created or changed to, (passing a number will used as an integer index).

\begin{verbatim}
1 \tblfrkv{my}{a,b,c,first=john,last=smith}\%
   [defaults={x=0,1=one,n=false,y=yes}]
2 \tblget{my.a}\\
3 \tblset{a}{{tRuE!!}}
4 \tblget{a}\\
5 \tblget{my.x}\\
6 \tblget{.x}\\
7 \tbladd{my.newkey}{val}\tblget{newkey}\\
8 \tbladd{nk}{VAL}\tblget{nk}\\
9 \tblif{n}{tr}{fa}\\
10 \tblif{my.y}{Tr}{FA}\\
11 \tblfrkvCD{M}{a=A,b=B,d=D}\[defaults={a,b,c,d}]
12 \dtblMa \dtblMb \dtblMc \dtblMd
\end{verbatim}

A practical tbl example
Right-angle sides $a=3$ and $b=4$ form a hypotenuse of $c=5$
Right-angle sides $a=3.20$ and $b=4.20$ form a hypotenuse of $c=5.28$
C: 5.28

Splitting strings

Splitting text (or a cmd) into oxford comma format via: \texttt{\textbackslash splittocomma} [expansion level]{text}{text to split on}:

\begin{verbatim}
\texttt{\textbackslash splittocomma}\{ j doe }{\{\textbackslash and\}-\textbackslash\}
\texttt{\textbackslash splittocomma}\{ j doe \textbackslash and s else }{\{\textbackslash and\}-\textbackslash}
\texttt{\textbackslash splittocomma}\{ j doe \textbackslash and s else \textbackslash and a per \} {\{\textbackslash and\}-\textbackslash}
\texttt{\textbackslash splittocomma}\{ j doe \textbackslash and s else \textbackslash and a per \textbackslash and f guy\} {\{\textbackslash and\}-\textbackslash}
\texttt{\textbackslash authors}\{ j doe \textbackslash and s else \textbackslash and a per \textbackslash and f guy\}
\texttt{\splittocomma}\{ \textbackslash authors\}\{\textbackslash and\}
\end{verbatim}

The expansion level is up to two characters, n|o|t|f, to control the expansion of each argument.

You can do a similar string split but to \texttt{\textbackslash item} instead of commas with \texttt{\splittoitems}
\begin{itemize}
\item kale
\item john
\item kale
\item john
\item someone else
\item 1
\item 2
\item 3
\item 4
\end{itemize}

PDF meta data (for pdflx package)

\writePDFmetadatakv *{m} Take a key-value string (eg. title=whatever, author=me) and writes to the jobname.xmpdata file, to be used by pdflx. * will first clear the data \writePDFmetadata runs the lua function penlight.tex.writePDFmetadata(), which pushes the lua variable __PDFmetadata__ (a table) to the xmpdata file.