penlightplus

Additions to the Penlight Lua Libraries

Kale Ewasiuk (kalekje@gmail.com)

2023–11–25

This package first loads the `[import]penlight` package.
The `pl` option may be passed to this package to create an alias for `penlight`.
`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:

```
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 gloals 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

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`

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

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

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

`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 sub directories

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.

`prtl(l),prtt(t)` print a literal string, or table

`wrt(x), wrtn(x)` write to log

`help_wrt(s1, s2)` pretty-print something to console. S2 is a flag to help you find., alias is `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 pre-defined (this uses `token.set_macro` internally)

`newcmd(cs, val)` like `\newcommand`

`renewcmd(cs, val)` like `\renewcommand`

`prvcmd(cs, val)` like `\providecommand`

`deccmd(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

global extras

If the package option `globals` is used, many additional globals are set for easier scripting. All `pl.tex` functions, and variables, `pl.hasval`, `pl.COMP`, `pl.utils.kpairs`, `pl.utils.npairs` become globals. `pl.tablex` is aliased as `TX` (which also includes all native Lua table functions), and `pl.array2d` is aliased as `A2d`.

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 insensitve) 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`)

```
\ifluax{3^3 == 27}{3*3*3 is 27}[WRONG]\\
\ifluax{abc123 == nil}{Var is nil}[WRONG]\\
\ifluax{not true}{tRuE}[fAlSe]\\
\ifluax{''}{TRUE}[FALSE]\\
\ifluaxv{''}{true}[false]\\
```

3*3*3 is 27
Var is nil
fAlSe
TRUE
false

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`
`\tblfrcsv` a shorthand `\tblfrkv {t}{csv}[naked_as_value=true,opts]`, a good way to convert a comma-separated list to an array
`\tblset {i}{v}` sets a value of the table/index `i` to `v`
`\tblget {i}` gets the value and `tex.sprint()`'s it
`\tbldef {i}{d}` pushes the value to a `cs` named `d`
`\tblgdef {i}{d}` pushes the value to a global
`\tbldefxy {i}{d}` splits the value of item by spaces creates two definitions `\dx` and `\dy` . Useful for pasing tikz coordinates like `xy=0 5`
`\iftbl {i}{tr}[fa]` runs code `ta` if the item is true else `fr`
`\iftblv {i}{tr}[fa]` runs code `ta` if the item is truthy else `fr`
`\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`.

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 allowered 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).

```
\tblfrkv{my}{a,b,c,first=john,last=smith}%
    [defaults={x=0,1=one,n=false,y=yes}]
\tblget{my.a}\\
\tblset{a}{tRuE!!}
\tblget{a}\\
\tblget{my.x}\\
\tblif{n}{tr}[fa]\\
\tblifv{n}{TR}[FA]\\
\tblif{my.y}{Tr}[Fa]\\
\tblifv{y}{tR}[fA]\\
%% \kvtblundefcheck % would throw error
\tbldef{my.first}{mydef} \mydef\\
{\tbldef{last}{mydef} \mydef} \mydef\\
{\tblgdef{last}{mydef}} \mydef\\

\tblset{my.a}{12 36}
\tbldefxy{my.a}{coord} (\coordx,\coordy)

\tblfrcsv{me}{a,b,"c,see",d,e}
\tblget{me/1},\tblget{2}\\
\tblget{3}\\
\tblset{me/4}{D}\tblget{me/4}\\
\tblset{5}{E}\tblget{5}\\
\tblget{-2},\tblget{me/-1}\\
%% \tblget{k} % would throw error
```

```
true
tRuE!!
0
fa
FA
Tr
tR
john
smith john
smith

(12,36)
a,b
c,see
D
E
D,E
```

world terre

Note: for thisf versions: all latex tbl commands are now prefixed with `tbl`, eg., `tblget`, `tblset`. Old-style commands eg. `gettbl` will be kept as aliases for a few more releases then removed.

Splitting strings

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

```
-\splittocomma{  j doe  }{\and}-\\
-\splittocomma{  j doe \and s else  }{\and}-\\
-\splittocomma{  j doe \and s else \and a per }{\and}-\\
-\splittocomma{  j doe \and s else \and a per \and f guy↩
   }{\and}-

\def\authors{j doe \and s else \and a per \and f guy}
\splittocomma[o]{\authors}{\and}
```

```
-j doe-
-j doe and s else-
-j doe, s else, and a per-
-j doe, s else, a per, and f
guy-
j doe, s else, a per, and f
guy
```

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 `\item` instead of commas with `\splittoitems`

```
\begin{itemize}
  \splittoitems{kale\and john}{\and}
  \splittoitems{kale -john -someone ↩
     else}{-}
  \splittoitems{1,2,3,4}{,}
\end{itemize}
```

- kale
- john
- kale
- john
- someone else
- 1
- 2
- 3
- 4