YAMLvars

a YAML variable parser for LuaLaTeX

Kale Ewasiuk (`kalekje@gmail.com`)

2023–07–18

YAMLvars is a LuaLaTeX-based package to help make definitions or produce LaTeX code using a YAML file. This package might be useful for you if you want to batch create documents by pushing various sets YAML data to a fixed LaTeX template, or just find it easier to read document metadata from a YAML file compared to the standard title, author, etc. commands.

1 Package Options

`useyv` By default, when you specify a YAML variable, it will be defined using `gdef` (only if it wasn't defined previously). If you use this setting, unless otherwise specified, YAML variables will be accessible under the `\yv{<var>}` command. This also allows numbers and symbols in the variable names. Note that internally, the variables are stored in the command sequence `yv--<var>`.

`parseCLI` If this option is enabled, any arguments passed to your lualatex compile command that end in ".yaml" will be used, separated by a space. If two yaml files are passed, the first one will be the declaration file, and the second will be the parsing file. They will be used at the beginning of the document. If one yaml file is passed, it will be treated as a parsing file, so you should declare the variables somewhere in the preamble. This option is offered to help with automation scripts. An example is showin in Section 8.

`allowundeclared` It might be helpful to define something in your YAML parsing doc without declaring it. If you want this flexibility, use this setting. Note that existing definitions will not be overwritten and an error will br thrown if the name exists. Alternatively, you can use the commands `\AllowUndeclaredYV` or `\ForbidUndeclaredYV` to toggle this behavior.

`overwritedefs` Danger! This will allow you to `gdef` commands with YAML. Caution should be taken to not set definitions like `begin`, `section`, etc.

2 Dependencies

`Note:` This package requires the `tinyyaml` package, available on CTAN.

The distribution: `https://github.com/api7/lua-tinyyaml`
`https://ctan.org/pkg/lua-tinyyaml`
The YAML specification: `https://yaml.org/spec/`

Many of the "transform" and "processing" functions built-in to this package rely on other packages, like `hyperref`, for example, but it is not loaded, and this package will only load `penlight`, `luacode`, `xspace`, and `etoolbox`.

3 Declaring variables

A declaration file can either be parsed with the command `declareYAMLvarsFile` command, or, if you want to do it LaTeX, you can put the YAML code in the `declareYAMLvars` environment. It is a declaring YAML document is (like all YAML) key-value dictionary: The top level key is the name of the variable to be defined/used. If the value of the top level is a string: it's interpreted as a single transform function to be applied. Otherwise, it must be a table that contains at least one of the following keys:
`xfm` (transform, may be a string or list of strings),
`prc` (processing, must be a single string), or
`dft` (default value, if being defined. Must be a string).

If you want to change the way a variable is initialized, you can change the function `YAMLvars.dec.PRC = function (var) ... end` where `PRC` is how the variable will be processed (`gdef`, `yvdef`, `length`, or something of your choosing).

The default value for variables is the Lua `nil`. YAMLvars will first check if the definition exists, if so, an error will be thrown so that we avoid overwriting. If the token is available, it is set to a package error, so that if the variable no defined later on, an error will tell the user they forgot to set it. This will be overwritten when you parse the variables and assign a value to it.

If you want a case-insensitive variable In the declaration YAML document, add a `lowcasevar: true` under the variable name. This will make the variable name lowercase before any transforms or processing is done. For example, if you have `title` as a YAML variable to set the `prc` function `setdocvar`, a user could write `Title` in the parsing file and still have it work. You can toggle this behaviour globally with the commands `\lowercasevarYVon` and `\lowercasevarYVoff` See the last example below.

You can change the default `xfm`, `prc`, or `dft` by changing the value (in Lua): `YAMLvars.xfmDefault = ''` etc.

Here is an example of a declaration document.

```
\begin{declareYAMLvars}
Location: addxspace                      # sets xfm=addxspace
People: [arrsortlastnameAZ, list2nl]     # BAD! don't do.
People:
  xfm: [arrsortlastnameAZ, list2nl]      # Correct way
Company:
  dft: Amazon                            # Change default only
Revisions:
  dft: '1 & \today & initial version \\'
  xfm: [sortZA, list2tab]
Rhead:
  prc: setRightHead


author:
  xfm: list2and     # (joins a list with \and (or lets a single string be passed)
  prc: setdocvar # calls \author{val}
  lowcasevar: true  # allows user to use Title: or TITLE:

title:
  xfm: lb2nl     # (make line-breaks \\)
  prc: setdocvar # calls \title{val}
  lowcasevar: true  # allows user to use Title: or TITLE:
\end{declareYAMLvars}
```

To change how a variable is declared (initialize), you can modify or add functions in `YAMLvars.dec` table, where the index is the same as the `prc` name. This function accepts two variables, the var name, and the default value set by dft. For lengths and toggles (from etoolbox), these functions are used to initialize lengths with newlength and newtoggle.

4 Parsing variables

A YAML file to be parsed will contain the variables as the top level keys, similar to declaring. The value can be anything you want; as long as you have applied appropriate transform and declaring functions to it so that it can be useful. For example, a value specified as a YAML list will first be interpreted as a Lua table (with numeric indexes/keys). You could declare a series of transforms functions to sort this table, map functions, and convert it to a series of LaTeX`\items`.

Here is an example of a parsing document.

```
\begin{parseYAMLvars}
Location: Planet Earth
People:                 # a YAML list
  - Some One            # turns into Lua table
  - No Body
# company assumed Amazon if not set here
Rhead: \today
\end{parseYAMLvars}
```

5 xfm – Transform Functions

These functions accept two arguments: `(var, val)` where `var` is the variable (or key) and val is the value. The transforms are specified as a list and are iteratively applied to the val. Usually, the final `xfm` function should produce a string so it can be defined.

Hint: if for some reason, your `xfm` and `prc` depends on other variables, you can access them within the function with `YAMLvars.varsvals`

5.1 Defining your own transform functions

After the package is loaded, you may add your function (somewhere in Lua) by adding it to the `YAMLvars.xfm` table. For example, if you wanted to wrap a variable's value with "xxx", here's how you could do that.

```
function myfunction(var, val)
        return 'xxx'..val..'xxx'
end
YAMLvars.xfm['addmyfunction'] = myfunction
```

If you want to run some Lua code and write in your YAML file (weird idea, but maybe useful for one-off functions), you can do so by specifying a transform function with an = in it to make a lambda function. For example, a `xfm` equal to "`= '---'..x..'---'`" would surround your YAML variable's value with em-dashes. You can access the variable name with this lambda function with `v`. If you want to just execute code (instead of settings `x =` , use `/`).

6 prc – Processing Functions

Like the transform functions, the processing function must accept `(var, val)`. Only one processing function is applied to the final (var, val) after the transforms are done.

This package includes `gdef` to set a definition, `yvdef` to define a variable under the `yv` command. `title, author, date` to set `\@title, \@author, \@date`, respectively

7 Some Examples

```
%! language = yaml
\begin{declareYAMLvars}
address:
  xfm:
    - list2nl
    - = x..'!!!'
name: null

title:
    xfm:
        - lb2nl
#         - / YAMLvars.prvcmd(titletext, YAMLvars.varsvals['atitle']:gsub('\n', ' ')..'\\xspace{}')
\end{declareYAMLvars}

%! language = yaml
\begin{parseYAMLvars}
title: |-
    A Multiline
    Monumental Title!

name: Joe Smith
address:
  - 1234 Fake St.
  - City
\end{parseYAMLvars}

\title

%\titletext!

\name

\address
```

A Multiline
Monumental Title!
Joe Smith
1234 Fake St.
City!!!

8 Automation Example

Suppose you had a number of bills of sales in yaml format and wanted to produce some nice pdfs. The following code shows how this could be done.

8.1 The main tex template

```
 %% main.tex
\documentclass{article}
\usepackage[paperheight=4in,paperwidth=3in,margin=0.25in]{geometry}
\usepackage[pl,func,extras]{penlight}
\usepackage[useyv,parseCLI]{YAMLvars}  % using command line option to make files
\usepackage{hyperref}
\usepackage{xspace}
\usepackage{luacode}

\setlength{\parindent}{0ex}
\setlength{\parskip}{0.75em}

\begin{luacode*} -- adding a custom function, put hfill between k-v pairs
    function YAMLvars.xfm.kv2hfill(var, val)
        local t = {}
        for k, v in pairs(val) do
            t[#t+1] = k..'\\hfill '..tostring(v)
        end
        return t
    end
\end{luacode*}


%! language = yaml
\begin{declareYAMLvars}
Customer: addxspace
Date: addxspace
Items:
    xfm: [kv2hfill, arr2itemize]
\end{declareYAMLvars}

\begin{document}
    Bill of sale for: \hfill \yv{Customer}\\
    Purchased: \hfill \yv{Date}\\
    \begin{itemize}
        \item[] ITEM \hfill PRICE
        \yv{Items}             % the yaml variable
        \begin{luacode*}
            totalcost = pl.tablex.reduce('+',
                pl.tablex.values(YAMLvars.varsvals['Items']), 0)
            tex.print('\\item[] TOTAL:\\hfill'..tostring(totalcost))
        \end{luacode*}
    \end{itemize}
```

```
\end{document}
```

8.2 The lua automation script

```
--automate.lua
for f in io.popen('dir .'):lines() do  -- get all files and info in cwd
    local i, j = f:find('%S*%.yaml')  --  find fnames
    if i ~= nil then
        f = f:sub(i,j) --  extract .yaml file name (no space in fname allowed)
        os.execute('lualatex -output-format=pdf main.tex '.. f)
                                    -- compile w/ yaml file as arg
        local fnew = f:gsub('yaml', 'pdf') -- file name for output pdf
        os.remove(fnew) -- delete if it exists already
        os.rename('main.pdf', fnew) -- change main.pdf to same as yaml file name
    end
end
```

8.3 The yaml data files

```
# sale1.yaml
Customer: Someone Cold
Date: January 2, 2021
Items:
    Toque: 12
    Mitts: 5.6
    Boots: 80

# sale2.yaml
Customer: Someone Warm
Date: July 1, 2021
Items:
    Beer (24 pk): 24
    Sunscreen: 5
    Hat: 12
```

9 xfm, dec, prc functions (from yamlvars.lua)

```lua
function YAMLvars.xfm.tab2arr(var, val)
     return pl.array2d.from_table(val)
end

function YAMLvars.xfm.arrsort2ZA(var, val)
    return pl.array2d.sortOP(val, pl.operator.strgt)
end

function YAMLvars.xfm.addrule2arr(var, val)
     return pl.array2d.map_slice2(_1..'\\\\\\'.. YAMLvars.setts.tabmidrule..' ', val, 1,-1,-2,-1)
end

function YAMLvars.xfm.arr2tabular(var, val)
     return pl.array2d.toTeX(val)..'\\\\'
end

function YAMLvars.xfm.list2items(var, val)
     return pl.List(val):map('\\item '.._1):join(' ')
end
YAMLvars.xfm.arr2itemize = YAMLvars.xfm.list2items

function YAMLvars.xfm.arrsortAZ(var, val)
     return pl.List(val):sort(pl.operator.strlt)
end

function YAMLvars.xfm.arrsortZA(var, val)
     return pl.List(val):sort(pl.operator.strgt)
end

local function complastname(a, b)
    a = a:split(' ')
    b = b:split(' ')
    a = a[#a]
    b = b[#b]
    return a < b
end

function YAMLvars.xfm.arrsortlastnameAZ(var, val)
    val = pl.List(val):sort(complastname)
    return val
end

function YAMLvars.xfm.list2nl(var, val)
    if type(val) == 'string' then
        return val
    end
    return pl.List(val):join('\\\\ ')
end

function YAMLvars.xfm.list2and(var, val) -- for doc vars like ↩
   author, publisher
    if type(val) == 'string' then
        return val
    end
    return pl.List(val):join('\\and ')
end


function YAMLvars.xfm.lb2nl(var, val) --linebreak in text 2 newline↩
    \\
    val, _ = val:gsub('\n','\\\\ ')
    return val
end

function YAMLvars.xfm.lb2newline(var, val) --linebreak in text 2 ↩
   newline \\
    val, _ = val:gsub('\n','\\newline ')
    return val
end

function YAMLvars.xfm.lb2par(var, val) --linebreak in text 2 new l
    val, _ = val:gsub('\n%s*\n','\\par ')
    return val
end

function YAMLvars.xfm.lowercase(var, val)
    return val:lower()
end


-- dec laration functions, -- -- -- -- -- -- -- -- -- -- -- -- -- ↩
   -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

function YAMLvars.dec.gdef(var, dft)
            YAMLvars.deccmd(var, dft)
end

function YAMLvars.dec.yvdef(var, dft)
        YAMLvars.deccmd('yv--'..var, dft)
end

function YAMLvars.dec.toggle(var, dft)
        tex.print('\\global\\newtoggle{'..var..'}')
        YAMLvars.prc.toggle(var, dft)
end
```

```

function YAMLvars.dec.length(var, dft)
        tex.print('\\global\\newlength{\\'..var..'}')
        YAMLvars.prc.length(var, dft)
end



-- prc functions (processing) -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

function YAMLvars.prc.gdef(var, val)
    --token.set_macro(var, val, 'global') -- old way, don't do as it will cause issues if val contains undef'd macros
    pl.tex.defcmd(var, val)
    YAMLvars.debugtalk(var..' = '..val, 'prc gdef')
end

function YAMLvars.prc.yvdef(var, val)
    pl.tex.defmacro('yv--'..var, val)
    YAMLvars.debugtalk('yv--'..var..' = '..val, 'prc yvdef')
end

function YAMLvars.prc.toggle(t, v) -- requires penlight extras
    local s = ''
    if pl.hasval(v) then
        s = '\\global\\toggletrue{'..t..'}'
    else
        s = '\\global\\togglefalse{'..t..'}'
    end
    tex.print(s)
    YAMLvars.debugtalk(s, 'prc toggle')
end

function YAMLvars.prc.length(t, v)
    v = v or '0pt'
    local s = '\\global\\setlength{\\global\\'..t..'}{'..v..'}'
    tex.print(s)
    YAMLvars.debugtalk(s, 'prc length')
end



function YAMLvars.prc.setATvar(var, val) -- set a @var directly: eg \gdef\@title{val}
    pl.tex.defcmdAT('@'..var, val)
end


```

```lua
function YAMLvars.prc.setdocvar(var, val) -- call a document var \var{val} = \title{val}
    -- YAML syntax options
    -- k: v -> \k{v}
    -- k:
    --   v1: v2      -> \k[v2]{v1}
    -- k: [v1, v2]   -> \k[v2]{v1}
    -- k: [v1]       -> \k{v1}
    if type(val) ~= 'table' then
        tex.sprint('\\'..var..'{'..val..'}')
    elseif #val == 0 then  -- assume single k,v passed
        for k,v in pairs(val) do
            tex.sprint('\\'..var..'['..v..']{'..k..'}')
        end
    elseif #val == 1 then
        tex.sprint('\\'..var..'{'..val[1]..'}')
    else
        tex.sprint('\\'..var..'['..val[2]..']{'..val[1]..'}')
    end
end


function YAMLvars.prc.setPDFdata(var, val)
    --update pdf meta data table (via penlight), uses pdfx xmpdata
    -- requires a table input
    for k, v in pairs(val) do
        if type(v) == 'table' then
            v = pl.List(v):join('\\sep ')
        end
        pl.tex.updatePDFtable(k, v, true)
    end
end

-- with hyperref package
function YAMLvars.prc.PDFtitle(var, val)
    tex.print('\\hypersetup{pdftitle={'..val..'}}')
end

function YAMLvars.prc.PDFauthor(var, val)
    tex.print('\\hypersetup{pdfauthor={'..val..'}}')
end

-- --


-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

function YAMLvars.makecmd(cs, val) -- provide command via lua
```