YAMLvars

a YAML variable parser for LuaLaTeX

Kale Ewasiuk (`kalekje@gmail.com`)

2021–11–07

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 docummnts 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 7.

llowundeclared 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 eisting definitions will not be overwritten and an error will br thrown if the name exists.

2 Dependencies

This package contains the `tinyyaml` Lua package. If you want to use it for other purposes, you can bring it into Lua by either:

```
\directlua{yaml = YAMLvars.yaml}            or
\directlua{yaml = require('tinyyaml')}
```

The distribution: `https://github.com/peposso/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. `penlight`, `xspace`, `hyperref` for example.

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 docuemnt is (like all YAML) key-value ditionary: 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).

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.

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
\end{declareYAMLvars}
```

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{declareYAMLvars}
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{declareYAMLvars}
```

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 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.

7.1 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}
```

7.2 The lua automation script

```lua
--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
```

7.3 The yaml data files

```yaml
# 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
```