Lua [placeholders]*

Erik Nijenhuis ⟨erik@xerdi.com⟩

23rd January, 2024

This file is maintained by **Xerdi**.
Bug reports can be opened at
`https://github.com/Xerdi/lua-placeholders`.

Abstract

A package for creating 'example' documents, which show parameters as placeholders and 'actual copy' documents, which show parameters with the real data, written in LuaTEX.

Contents

*This document corresponds to `lua-placeholders` version 1.0.0, written on 2024-01-23

1 Introduction

This package is meant for setting parameters in a LuaLaTeX document in a more programmatically way with YAML. Parameters can be specified by adding a 'recipe' file. These recipe files describe the parameter's type, placeholders and/or default values. From thereon, the placeholders can be displayed in the document and an '*example*' document can be created. An '*actual copy*' document can be created by loading additional payload files, which all must correspond to a recipe file.

1.1 Pros

1. Create an '*example*' or '*actual copy*' document with the same LaTeX source and YAML recipe.
2. Integration within systems is as easy as compiling a normal LaTeX document, especially thanks to the fallback support to JSON, which is quite renown in programming languages.
3. Supports multiple data types and formatting macros which work in most TeX environments, like `enumerate` or `tabular`.

1.2 Cons

1. The package only works with LuaLaTeX.
2. In order for the files to be loaded, commandline option '`--shell-escape`' is required.

1.3 Prerequisites

If you're using JSON as ⟨*recipe*⟩ and ⟨*payload*⟩ format, the following requirements are no longer needed, since LuaTeX already supports JSON formats out of the box.

For YAML support, however, this package requires the `lyaml`[1] Lua module for parsing the YAML files. This also includes the `libYAML`[2] platform dependent library and optionally LuaRocks for installing `lyaml`. Another requirement is Lua, which version meets the Lua version used by LuaTeX. If no `LUA_PATH` is set, and you use LuaRocks, this package tries to call the LuaRocks executable to find the `LUA_PATH`. If `lyaml` can't be loaded, this package will first try to fall back with `lua-tinyyaml`[3] for lesser YAML support and secondly fall back on accepting JSON files only.

2 Usage

This section describes the basic commands of `lua-placeholders`. For more detail about type specific commands or the behavior of types with commands described here, see section 3.

2.1 Configuration

`\strictparams` In order to give an error when values are missing, the `\strictparams`[1] command can be used. Make sure to do it before loading any ⟨*recipe*⟩ and ⟨*payload*⟩ `\loadrecipe` files. In order to load a recipe the macro `\loadrecipe[`⟨*namespace*⟩`]{`⟨*filename*⟩`}` can be used. Where the ⟨*filename*⟩ is a YAML file with its corresponding extension. The optional ⟨*namespace*⟩ is only a placeholder in order to prevent any conflicts between duplicate ⟨*key*⟩s. If left out, the ⟨*namespace*⟩ de- `\loadpayload` faults to the base name of the filename. The same behaviour counts for `\loadpayload[`⟨*namespace*⟩`]{`⟨*filename*⟩`}`. The order of loading ⟨*recipe*⟩ and ⟨*payload*⟩ files doesn't matter. If the ⟨*payload*⟩ file got loaded first, it will be yielded until the corresponding ⟨*recipe*⟩ file is loaded.

All other macros of this package also take the optional ⟨*namespace*⟩, which `\setnamespace` by default is equal to `\jobname`. This default ⟨*namespace*⟩ can be changed with `\setnamespace{`⟨*new default namespace*⟩`}`.

2.2 Displaying Parameters

For displaying variables, the commands `\param` and `\PARAM` share the same inter- `\param` face. The most trivial, displaying the variable as-is, is `\param[`⟨*namespace*⟩`]{`⟨*key*⟩`}`. `\PARAM` The `\PARAM` however, shows the value as upper case.

In some cases, it's required to output the text without any TEX related functionality. Another case is that some environments don't take macros with optional `\rawparam` arguments well. For these cases there is `\rawparam{`⟨*namespace*⟩`}{`⟨*key*⟩`}`, which takes the namespace as mandatory argument, instead of optional, and doesn't output fancy TEX placeholders.

`\hasparam` To check whether a parameter is set, the `\hasparam[`⟨*namespace*⟩`]{`⟨*key*⟩`}{`⟨*true case*⟩`}{`⟨*false case*⟩`}` command is used. However, a more robust way is using LATEX hooks. For recipes being loaded, the hook `namespace/`⟨*name*⟩ is triggered once. For payloads being loaded, the hook `namespace/`⟨*name*⟩`loaded` is triggered once. For more information on LATEX hooks, read the `lthooks` manual.

[1] The `\strictparams` command is still under development.

3 Parameter Specification

Every parameter specified has a ⟨*type*⟩ set. Optionally there is a choice between setting a ⟨*default*⟩ or a ⟨*placeholder*⟩ for the parameter.

bool Next to the textual representation of *true* and *false*, it provides a LaTeX command using the `ifthen` package. Therefore, only the ⟨*default*⟩ setting makes sense.

Recipe

```
bool example:
  type: bool
  default: false
```

Payload

```
bool example: true
```

`\param` `\ifparam` With a boolean type the `\param[`⟨*namespace*⟩`]{`⟨*name*⟩`}` returns either *true* or *false*. Additionally, it provides the `\ifparam[`⟨*namespace*⟩`]{`⟨*name*⟩`}{`⟨*true code*⟩`}{`⟨*false code*⟩`}` command for top level boolean types. The macro is just a wrapper for the boolean package `ifthen`, which supports spaces in names.

string representing a piece of text. All TeX related symbols in the text, like \, % and #, are escaped.

```
string example:
  type: string
  placeholder: A string
```

```
string example: PeelInc.
```

`\param` A string type can easily be placed in LaTeX using the `\param` command.

number representing a number, like the number type of Lua. In most cases it's necessary to use ⟨*default*⟩ instead of ⟨*placeholder*⟩, especially when the number is used in calculations, since a placeholder will cause errors in LaTeX calculations.

```
number example:
  type: number
  default: -1
```

```
number example: 1
```

`\numparam` A number type can be used with `\param`, just like the string type. However, the `\numparam[`⟨*namespace*⟩`]{`⟨*name*⟩`}` command uses `\numprint` to properly format the number according to the selected language. Read the documentation of package `numprint` for more information.

list representing a list of values. The value type is specified by ⟨*value type*⟩. A ⟨*default*⟩ setting can be set. Due to its structure, a ⟨*placeholder*⟩ would be somewhat incompatible with the corresponding macros. However, a placeholder can be simulated by setting the placeholders as children of the ⟨*default*⟩ list, as demonstrated in the example.

```
10 list example:
11   type: list
12   item type: string
13   default:
14     - A string
15     - A second string
```

```
4 list example:
5   - Tomatoes
6   - Potatoes
```

`\param` `\paramlistconjunction` Command `\param` concatenates every item with command `\paramlistconjunction`. By default, the conjunction is set to '`,~`'.

`\forlistitem` There's also the `\forlistitem[`⟨*namespace*⟩`]{`⟨*name*⟩`}{`⟨*csname*⟩`}` command, which takes an additional ⟨*csname*⟩ and will execute it for every item in the list. This command doesn't handle advanced features like altering the conjunction. Though, some utility commands will be set, which are only available in the ⟨*csname*⟩s implementation, in order to achieve the same goal.

object representing a list of key value pairs. This parameter type requires a ⟨*fields*⟩ specification to be set. Any field must be of type `bool`, `number` or `string`.

```
16 object example:
17   type: object
18   fields:
19     name:
20       type: string
21       placeholder: Your name
22     email:
23       type: string
24       placeholder: Your email
```

```
7 object example:
8   name: John Doe
9   email: j.doe@example.com
```

`\paramfield` There is no support for the `\param` command. In order to show to contents there is the `\paramfield[`⟨*namespace*⟩`]{`⟨*name*⟩`}{`⟨*field*⟩`}` command. However, unlike the common command `\param`, the command `\hasparam` does work with object types.

`paramobject` There's also the `paramobject` environment, which takes an optional ⟨*namespace*⟩ and takes the ⟨*name*⟩ of the object as arguments and then defines for every field name a corresponding command. Every command is appended with the `\xspace` command to prevent gobbling a space. In other words, the author doesn't have to end the command with accolades '{}' to get the expected output.

table representing a table. This parameter type requires a ⟨*columns*⟩ specification to be set. The ⟨*columns*⟩ describes each column by name with its own type specification. Like the object field, only the types `bool`, `number` and `string` are supported column types.

```
table example:
  type: table
  columns:
    description:
      type: string
      placeholder: The
          description
    price:
      type: number
      placeholder: The price
```

```
table example:
  - description: Peeling
      tomatoes
    price: 50.00
  - description: Peeling
      potatoes
    price: 25.00
```

`\fortablerow`

Like the object, the table has no support for `\param`, but comes with a table specific command `\fortablerow[`⟨*namespace*⟩`]{`⟨*name*⟩`}{`⟨*csname*⟩`}`. The control sequence name ⟨*csname*⟩ is a user-defined command with no arguments, containing any of the column names in a command form. For example, the name `example` would be accessible as `\example` in the user-defined command body.

Like the object field, a table cell doesn't require accolades, though, this is due to the Lua implementation behind it. Technically every command in the user-defined command body is replaced with the variable in Lua, instead of redefining the command itself for every row, preventing issues with macro expansion between table rows and also column separators in TEX.

4 References

[1] Andrew Danforth. *lyaml*. `https://github.com/gvvaughan/lyaml` and `https://luarocks.org/modules/gvvaughan/lyaml`. Accessed: 6 January, 2024.

[2] *libYAML*. `https://pyyaml.org/wiki/LibYAML` and `https://packages.msys2.org/package/mingw-w64-x86_64-libyaml`. Accessed: 6 January, 2024.

[3] *lua-tinyyaml*. `https://ctan.org/pkg/lua-tinyyaml` and `https://github.com/api7/lua-tinyyaml`. Accessed: 23 January, 2024.

5 Example

The source file `example.tex` is a perfect demonstration of all macros in action. It shows perfectly what happens when there's a ⟨*payload*⟩ file loaded and when not.

The result of this example  is attached in the digital version of this document.

Listing 1: `example.tex`

```
''
\documentclass{article}
\usepackage{gitinfo-lua}
\usepackage{lua-placeholders}
\usepackage{listings}
\usepackage{amsmath}
\usepackage{calc}

\loadrecipe[\jobname]{example-specification.yaml}

\setlength{\parindent}{0pt}

\begin{document}
    \title{Lua \paramplaceholder{placeholders} Example\thanks{This
        example corresponds to \texttt{lua-placeholders} version \
        gitversion{} written on \gitdate.}}
    \author{\dogitauthors[\\]}
    \maketitle

    \section*{Basics}
    Wrong parameter:\\

    \lstinline[style=TeX,morekeywords={param}]|\param{non existing}|
    $\implies$
    \param{non existing}\\

    Conditional Parameter:\\

    \lstinline[style=TeX,morekeywords={hasparam}]|\hasparam{list
        example}{is set}{is not set}|
    $\implies$
    \hasparam{list example}{is set}{is not set}

    \section*{Before values loaded}

    Boolean example:\\

    \lstinline[style=TeX,morekeywords={param}]|\param{bool example}|
```

```
    $\implies$
    \param{bool example}\\

    \lstinline[style=TeX,morekeywords={ifparam}]|\ifparam{bool
        example}{TRUE}{FALSE}|
    $\implies$
    \ifparam{bool example}{TRUE}{FALSE}\\

    String example:\\

    \lstinline[style=TeX,morekeywords={param}]|\param{string example
        }|
    $\implies$
    ``\param{string example}''\\

    Number example:\\

    \lstinline[style=TeX,morekeywords={param}]|\param{number example
        }|
    $\implies$
    \param{number example}\\

    List example:\\

    \lstinline[style=TeX,morekeywords={param}]|\param{list example}|
    $\implies$
    \param{list example}\\

    \begin{lstlisting}[language={[LaTeX]TeX},morekeywords={
        formatitem,forlistitem}]
\begin{enumerate}
    \newcommand\formatitem[1]{\item #1}
    \forlistitem{list example}{formatitem}
\end{enumerate}
    \end{lstlisting}
    $\implies$
    \begin{enumerate}
        \newcommand\formatitem[1]{\item #1}
        \forlistitem{list example}{formatitem}
    \end{enumerate}

    Object example:\\

    \lstinline[style=TeX,morekeywords={paramfield}]|\paramfield{
        object example}{name}|\\
    \lstinline[style=TeX,morekeywords={paramfield}]|\paramfield{
```

```
        object example}{email}|\\
96      $\implies$
97      \paramfield{object example}{name}
98      \paramfield{object example}{email}\\
99
100     \begin{lstlisting}[style=TeX,morekeywords={name,email}]
101 \newcommand\name{...}
102 \begin{paramobject}{object example}
103     \name \email
104 \end{paramobject}
105 % And here it works again
106 \name
107     \end{lstlisting}
108     $\implies$
109     \newcommand\name{...}%
110     \parbox{\linewidth}{
111     \begin{paramobject}{object example}
112         \name \email
113     \end{paramobject}
114     \name
115     }\\
116
117     Table example:\\
118
119     \begin{lstlisting}[style=TeX,morekeywords={formatrow,fortablerow
            ,description,price}]
120 \newcommand\formatrow{\description & \price \\}%
121 \begin{tabular}{l | l}
122     \textbf{Description} & \textbf{Price} \\ \hline
123     \fortablerow{table example}{formatrow}
124 \end{tabular}
125     \end{lstlisting}
126     $\implies$
127     \newcommand\formatrow{\description & \price \\}%
128     \begin{tabular}{l | l}
129         \textbf{Description} & \textbf{Price} \\ \hline
130         \fortablerow{table example}{formatrow}
131     \end{tabular}
132
133
134     \section*{After values loaded}
135     \loadpayload[\jobname]{example.yaml}
136
137     Boolean example:\\
138
139     \lstinline[style=TeX,morekeywords={param}]|\param{bool example}|
```

```
    $\implies$
    \param{bool example}\\

    \lstinline[style=TeX,morekeywords={ifparam}]|\ifparam{bool
        example}{TRUE}{FALSE}|
    $\implies$
    \ifparam{bool example}{TRUE}{FALSE}\\

    String example:\\

    \lstinline[style=TeX,morekeywords={param}]|\param{string example
        }|
    $\implies$
    ``\param{string example}''\\

    Number example:\\

    \lstinline[style=TeX,morekeywords={param}]|\param{number example
        }|
    $\implies$
    \param{number example}\\

    List example:\\

    \lstinline[style=TeX,morekeywords={param}]|\param{list example}|
    $\implies$
    \param{list example}\\

    \begin{lstlisting}[language={[LaTeX]TeX},morekeywords={
        formatitem,forlistitem}]
\begin{enumerate}
    \newcommand\formatitem[1]{\item #1}
    \forlistitem{list example}{formatitem}
\end{enumerate}
    \end{lstlisting}
    $\implies$
    \begin{enumerate}
        \newcommand\formatitem[1]{\item #1}
        \forlistitem{list example}{formatitem}
    \end{enumerate}

    Object example:\\

    \lstinline[style=TeX,morekeywords={paramfield}]|\paramfield{
        object example}{name}|\\
    \lstinline[style=TeX,morekeywords={paramfield}]|\paramfield{
```

```
        object example}{email}|\\
    $\implies$
    \paramfield{object example}{name}
    \paramfield{object example}{email}\\

    \begin{lstlisting}[style=TeX,morekeywords={name,email}]
\newcommand\name{...}
\begin{paramobject}{object example}
    \name \email
\end{paramobject}
% And here it works again
\name
    \end{lstlisting}
    $\implies$
    \parbox{\linewidth}{
    \begin{paramobject}{object example}
        \name \email
    \end{paramobject}
    \name
    }\\

    Table example:\\

    \begin{lstlisting}[style=TeX,morekeywords={formatrow,fortablerow
        ,description,price}]
\newcommand\formatrow{\description & \price \\}%
\begin{tabular}{l | l}
    \textbf{Description} & \textbf{Price} \\ \hline
    \fortablerow{table example}{formatrow}
\end{tabular}
    \end{lstlisting}
    $\implies$
    \begin{tabular}{l | l}
        \textbf{Description} & \textbf{Price} \\ \hline
        \fortablerow{table example}{formatrow}
    \end{tabular}

    \section*{Specification File}
    \lstinputlisting[language=YAML,numbers=left,xleftmargin=15pt,
        caption={example-specification.yaml},columns=fullflexible]{
        example-specification.yaml}

    \clearpage

    \section*{Payload File}
    \lstinputlisting[language=YAML,numbers=left,xleftmargin={15pt},
```

```
        caption={example.yaml},columns=fullflexible]{example.yaml}

    \section*{Fallback Payload}
    \lstinputlisting[language=JSON,numbers=left,xleftmargin={15pt},
        caption={example.json},columns=fullflexible]{example.json}
\end{document}
```